ICGST-AIML Journal, ISSN: 1687-4846, Volume 8, Issue II, September 2008

www.icgst.com

Employing four ANNs Paradigms for Software Reliability Prediction: an Analytical Study

Sultan H. Aljahdali[1] and Khalid A. Buragga[2]

[1]College of Computers and Information Systems, Taif University, Taif, Saudi Arabia
[2]College of Computer Sciences & I.T., King Faisal University, Hofuf, Saudi Arabia
Email: aljahdali@tu.edu.sa and kburagga@kfu.edu.sa

Abstract

Software Reliability is a key concern of many users and developers of software. Demand for high software reliability requires robust modeling techniques for software quality prediction. Software reliability models are very useful to estimate the probability of the software fail along the time. Several different models have been proposed to predict the software reliability growth model (SRGM); however, none of them has proven to perform well considering different project characteristics. The ability to predict the number of faults in the software during development and testing processes. In this paper, we explore connectionist artificial neural networks models as an alternative approach to derive these models by investigating the performance analysis of four different connectionist paradigms for modeling the software reliability prediction. The presented four paradigms are multi-layer perceptron neural network, radial-basis functions, Elman recurrent neural networks and a Takagi-Sugeno fuzzy inference system learned using a neural network algorithm (neuro-fuzzy model). The results show that the neural network model adopted has good predictive capability.

Keywords: *Neural Network, Software Quality, Software Reliability, and Time-Series Prediction*

1. Introduction

Software reliability is becoming more and more important in software industry various techniques are required to discover faults in the development of software. Software reliability can defined as the probability of a software system to perform its specified functions correctly over a long period of time or for different input set under the usage environments similar to that of its target customer [22]. However; as reliability of software is measured in terms of failure it is impossible to measure reliability before the software is developed completely, software reliability is the most extensively studied quality among all the quality attributes [17].

In the past few years a number of software reliability assessment models have been developed to solve software reliability models. These software models have been developed in response to the urgent need for software engineers, system engineers and managers to quantify the concept of software quality prediction. The information provided by the models is helpful in making management decisions on issues regarding the software reliability. The software reliability models can be grouped into two categories:

1. Analytical software reliability growth models (SRGM) and;
2. Data-driven models.

The analytical SRGMs use stochastic models to describe the software failure process under several assumptions to provide mathematical tractability [9]. The major drawbacks of these models are their restrictive assumptions. On the other hand, most data-driven models follow the approach of time series analysis, including traditional autoregressive methods [1, 5, 14, 20] and modern artificial neural network (ANN) techniques [12], Case based reasoning, and optimal set reduction and genetic algorithms [7, 2, 16]. These models are developed from past software failure history data. The main objective of these models are to help predict which modules are error prone which in turn can help developer to focus on many aspects of maintenance cycle [3, 4]. The connectionist neural networks are excellent forecasting tools and can learn from scratch by adjusting the interconnections between layers. It contains many models and learning algorithms offer excellent learning capability based on statistical learning theory [1, 13]. Moreover, fuzzy inference systems are excellent for decision making under uncertainty can be integrated within this framework to get the neuro-fuzzy model which is used to fine-tune the parameters of fuzzy inference systems [1].

The purpose of this paper is to investigate the performance analysis of four different connectionist paradigms for modeling the software reliability prediction. The four different techniques considered are multi-layer perceptron neural network trained using the back-propagation, radial-basis functions, Elman recurrent neural networks and a Takagi-Sugeno fuzzy inference system learned using a neural network algorithm (neuro-fuzzy model).

The rest of the paper is organized in the following manner. Section 2, presents an overview of the different connectionist models and their learning methodologies. In section 3, the data set used in this study is described briefly. In section 4, we provide an experimental setup and performance evaluation. Detailed experiments results are provided in section 5. Finally, section 6 concludes the paper.

2. Connectionist Predicting Models

In the past three decades, hundreds of models were introduced to estimate the reliability of software systems. The issue of building growth models was the subject of many research works which helps in estimating the reliability of a software system before its release to the market. There are appearing two major trends in software reliability research: the use of analytical software reliability growth models (SRGM) and data-driven models. In this paper, the data-driven models using connectionist models. The connectionist models "learn" by adjusting the interconnections between layers. When the network is adequately trained, it is able to generalize relevant output for a set of input data. Learning typically occurs by example through training, where the training algorithm iteratively adjusts the connection weights (synapses). In an artificial neural network learning occurs by the iterative updating of connection weights using learning algorithms. The ANN methodology enables us to design useful nonlinear systems accepting large numbers of inputs, with the design based solely on instances of input-output relationships. For a training set T consisting of n argument value pairs and given a d-dimensional argument x and an associated target value t will be approximated by the neural network output. The function approximation could be represented as

$$g : X \rightarrow T \tag{1}$$

In most applications the training set T is considered to be noisy and our goal is not to reproduce it exactly but rather to construct a network function that generalizes well to new function values. We will try to address the problem of selecting the weights to learn the training set. The notion of closeness on the training set T is typically formalized through an error function of the form

$$E = \sum_{i=1}^{n} (y_i - t_i)^2 \tag{2}$$

where y_i is the network output.

2.1 Multi-Layer Perceptron

A multilayer perceptron (MLP) is shown in Figure 1 is the most widely used feed-forward neural networks and one of the most popular training algorithms for feed-forward neural networks is the back-propagation algorithm. The back-propagation learning algorithm is developed for multilayer perceptrons is a form of gradient descent. As such, it is vulnerable to local minima in weight space. A number of techniques for forcing back-propagation networks out of a local minimum have been proposed in the literature. One of the simplest is adding a momentum term to the back-propagation formula, which adds an additional vector to the current weight update. Other schemes include the Newton's method, or conjugate gradient algorithms [8]. As shown in Figure 1, the feed-forward neural network architecture consists of a number of elements called neurons. These neurons are grouped together to form a layer. Each neuron has a number of inputs and a single output. Each input has an assigned factor or parameter called the weight.

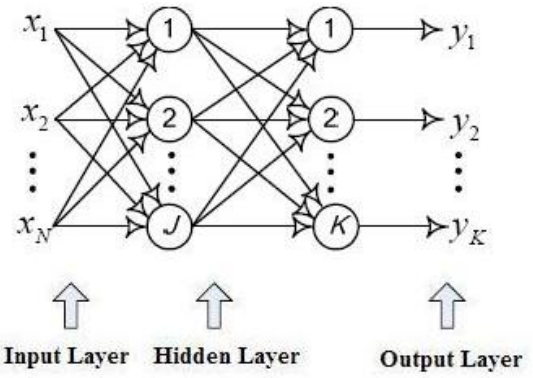

Figure 1, the multi-layer perceptron neural networks structure

The back-propagation training algorithm is an iterative gradient descent algorithm designed to minimize the mean square error E between the actual output of a multilayer feed forward perceptron and the desired output and updates the weights by moving them along the gradient-descendent direction. This can be summarized with the expression

$$\Delta w = -\eta \nabla E \tag{3}$$

where the parameter $\eta > 0$ is the learning rate that controls the learning speed. From Equation (3) it is clear that the error function to be minimized is a highly nonlinear high dimensional surface (the dimension given by the number of weights in the network) with possibly many local minima. Though a gradient descent algorithm is fast, it does not guarantee that the minimum found is the global one.

2.2. Radial Basis Function Neural Network

Radial Basis functions are used for strict interpolation in a multi dimensional space. Radial Basis networks [8, 6] typically have two distinct layers as shown in Figure 2. Each layer is fully connected to the following one and the hidden layer is composed of a number of nodes with radial activation functions called radial basis functions each of which is a function of the distance between an input pattern and a prototype pattern. Each of the input components feeds forward to the radial functions. The outputs of these functions are linearly combined with weights into the network output. Each radial function has a local response (opposite to the global response of *sigmoid function*) since their output only depends on the distance of the input from a center point. A standard choice of basis function is the Gaussian:

$$\phi_j(x) = exp\left\{-\frac{\|x-\mu_j\|^2}{2\sigma^2}\right\} \quad (4)$$

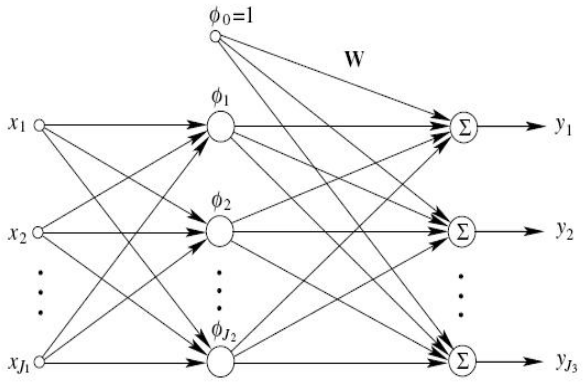

Figure 2. RBFN configuration

where x is an input pattern, ϕ_j is a radial basis function centered at location μ_j .The right layer is a simple linear discriminate that outputs a weighted sum of the basis functions. The equation for a single output y_k is:

$$y_k(x) = \sum_{j=1}^{J_2} w_{kj}\phi_j(x) + w_{k0} \quad (5)$$

The hidden layer performs a nonlinear transform of the input, and the output layer is a linear combiner mapping the nonlinearity into a new space. The biases of the output layer neurons can be modeled by an additional neuron in the hidden layer, which has a constant activation function $\phi_j(x) = 1$. The degree of accuracy of these RBF networks can be controlled by three parameters: the number of basis functions used, their location and their width. The training of the RBF network is radically different from the classical training of standard MLP. In RBFN networks with the chosen type of radial basis function, training resolves itself into selecting the centers and dimensions of the functions and calculating the weights of the output neuron. The number of hidden layer neurons can be determined through iteratively adding neurons until the network error falls under a certain training error goal. Radial basis transfer functions by nature tend to cover a local portion of the input spectrum thus having better focus on local details.

2.3. Elman Recurrent Neural Networks

In feed-forward neural networks there are no feedbacks from the output of one layer to the inputs of the same layer (i.e. no interconnection between the nodes within the same layer) or earlier layers of nodes. Also, these networks have no *memory* (i.e. the input vector at any time instant determines the output, assuming the weights do not vary). A recurrent neural network is different from the feed-forward neural network because it has feedback connections. Similar to the use of feedback in control systems, recurrent neural networks take into consideration the dynamic behavior of systems. The output of a node at any time instant t depends on its inputs at time instant t and those feedback connections whose values are a time instant earlier $(t-\Delta t)$, where Δt is the sampling time. As the current output of the recurrent neural network depends on both current and prior inputs, recurrent networks function behave just like memories which have stored values. Elman neural network [8], is a partial recurrent network model which has a number of *context nodes* in the input layer as shown in Figure 3. The context nodes do nothing more than duplicate the activity of a hidden layer, at the previous time step, at the input of the network. The Elman network commonly is a two-layer network with feedback from the first-layer output to the first layer input.

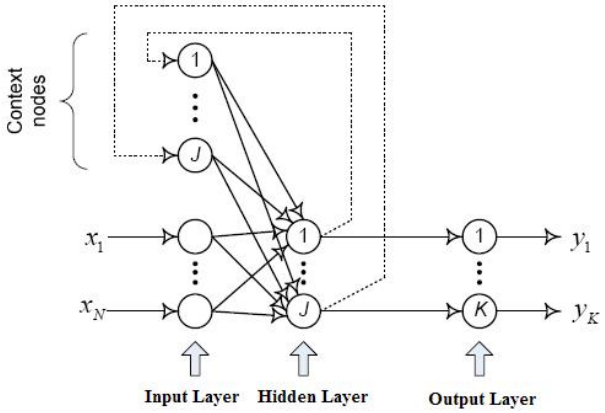

Figure 3. Structure of Elman recurrent neural network

The advantage of Elman networks over fully recurrent networks is that back propagation [23] is used to train the network while this is not possible with other recurrent networks where the training algorithms are more complex and therefore slower.

2.4 Fuzzy Neural Network

Neuro-fuzzy models are hybrids of artificial neural networks and fuzzy logic. The adaptive network based fuzzy inference system (ANFIS) is a fuzzy inference system executed in an adaptive network and it can establish an input-output relation through the back-propagation process with an artificial intelligence style (if-then rules of fuzzy inference). ANFIS is a hybrid of two intelligent system models. It combines the low-level computational power of a neural network with the high-level reasoning capability of a fuzzy inference system. In the aspect of modeling, ANFIS can easily establish non-linear functions and it can forecast time sequence of no qualitative relations. The easiest way to understand how the ANFIS model operates is to consider it in two steps. First, the system is trained in a similar way to a neural network with a large set of input data. Then, once trained, the system operates exactly as a fuzzy expert system. The ANFIS are fuzzy Sugeno models put in the framework of adaptive systems to facilitate learning and adaptation [11]. Such framework makes such models systematic and less relying on expert knowledge. To describe the ANFIS architecture briefly, consider two fuzzy if-then rules based on a first order Sugeno model:

Rule 1: IF (x is A_1) and (y is B_1) **THEN** ($f_1 = p_1 x + q_1 y + r_1$)
Rule 2: IF (x is A_2) and (y is B_2) **THEN** ($f_2 = p_2 x + q_2 y + r_2$)

where x and y are inputs; A_i and B_i are appropriate fuzzy sets; p_1, q_1, and r_1 are certain parameters. f_1 and f_2 contribute to the output of the system. The fuzzy inference system has, for example, two inputs, one output, five layers of framework (shown in Figure 4), and two learning stages. In the first layer (input layer), the input variables are mapped into fuzzy sets to estimate their degrees of membership by the designated membership functions. At the second layer (rule layer), the perquisite conditions of fuzzy logic rules are matched with input variables in order to obtain the weights, i.e., firing strength, of the rules which are the multiplication results of all inputs by using T-norm multiplication operation. At the normalization layer (the third layer), the relative ratios of weights of all rules are calculated for the nodes in this layer. Then, the relative weights are multiplied by the functions of factor sets at the conclusion inference layer (the fourth layer). At the last and fifth layer (output layer), all the information from the previous layer is aggregated to calculate output variable, just like the procedure of defuzzification. From the calculation of the five layers, it is clear that the function of ANFIS is similar to that of Sugeno model.

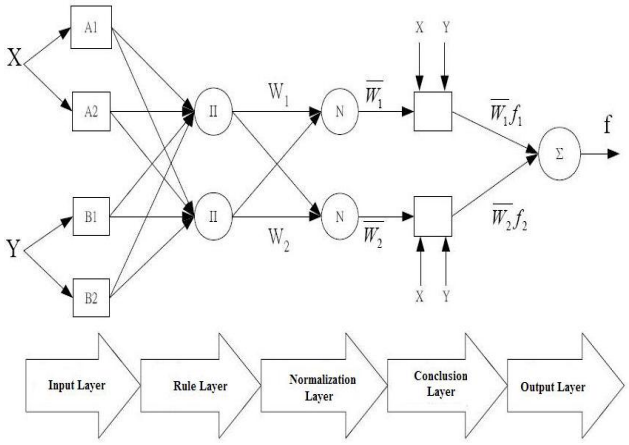

Figure 4.The framework of adaptive network based fuzzy inference system (ANFIS).

ANFIS learns using a hybrid learning algorithm combining a least-squares estimator and the gradient descent method [10]. Firstly, initial activation functions are randomly assigned to each neuron such that the range is spread across the domain of the input and the widths of the fuzzy sets are large enough to allow suitable overlap. Then the training begins and each epoch (training run) consists of a forward pass *and* a backward pass, updating the synaptic links according to the input data and desired result. The forward pass adjusts the neuron consequents, layer-by-layer, to minimize the error. Once it has reached the output of the last layer, the backward pass begins and the antecedents are updated as the consequents are held constant.

3. Data Set

The DACS Services at the Department of Defense (D.O.D.) Software Information Clearinghouse provides an authoritative source for the state of the art software information, supplying technical support for the software community. John Musa of Bell Telephone Laboratories compiled a software reliability database. His objective was to collect failure interval data to assist software managers in monitoring test status, predicting schedules and to assist software researchers in validating software reliability models [18]. These models are applied in the discipline of Software Reliability Engineering. The dataset consists of software failure data on 16 projects. Careful controls were employed during data collection to ensure that the data would be of high quality. The data was collected throughout the mid 1970s. It represents projects from a variety of applications including real time command and control, word processing, commercial, and military applications. In our paper, we used data from one project, the .Real-Time Control project.

4. Experiment Setup

4.1. Test/Debug data for Real-Time Control

Observation of data for test/debug of a program for real-time control was used. The size of the program is 870 kilo-steps of FORTRAN and a middle level language. Since the test data is recorded day by day, the test operations performed in a day are regarded to be a test instance. The trainings accomplish for different models by dividing the data set into two sections, training and test sets, comprising of 70% and 30% of the total data set respectively. For real-time and control case study, we took the first 96 data points for training and the next 40 points for prediction/ validation of the developed model.

4.2. Connectionist Models Structures

We implemented the connectionist models using MATLAB where the architecture of the MLP neural networks used for software reliability prediction consists of an input layer, one hidden layer, and an output layer. The input layer contains a number of neurons equal to the number of delayed measurements allowed to build neural networks model. In our case, there are four inputs to the network, they are $y(k-1)$, $y(k-2)$, $y(k-3)$, and $y(k-4)$. Where $y(k-1)$ the observed faults one-day is before the current day, the hidden layer consists of 4 nodes. The output layer consists of one output neuron producing the estimated value of the fault. The hidden layer and output layer nodes have tanh-sigmoidal activation function and learning rate is 0.002. The structure of Elman network has the same architecture of MLP. Similarly, the RBFN has the same architecture with σ = 4000 and hidden layer activation functions are Gaussian function and output layer nodes have linear activation functions. The structure of the ANFIS model consists of a Sugeno type fuzzy system with generalized bell input membership functions and a linear output membership function. The network consists of 4 inputs, each with 3 input membership functions, 12 rules and 1 output membership function. The training for these models was terminated after 20 epochs.

5.3. Evaluation Criteria

We used an evaluation criterion for each developed model to measure its performance. The criterion of evaluation (i.e. performance) to measure the performance of the developed connectionist model was defined as the sum of the square of the error:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(y(k) - \tilde{y}(k))^2} \quad (6)$$

Where $y(k)$ is the observed fault and $\tilde{y}(k)$ is the predicted fault for the given model structure and N represents the number of measurements used for estimating the model.

5. Experimental Results

This section demonstrated the results of the different connectionist learning algorithms runs, which explore various aspects of specification connectionist modeling methodology For each learning algorithms run, the best model is evaluated over the training and test set using prediction performance measurements. The training data from real time control and their predicted results from different model are shown in Figure 5 and the predicted absolute error in Figure 6. The forecasted and actually measured values where compared to verify the generated models and their predictability and generalization capability in Table 1.

Table 1- the comparison among different connectionist models for training/test data set

	RMSE	
	Training Data	Test Data
MLP	0.6061	0.6677
RBFN	1.6465	0.1591
Elman	0.1625	0.1394
ANFIS	1.3364	0.9079

As shown in Table 1, the recurrent network give the minimum error in training and test data set due taking into consideration the dynamic behavior of systems. The MLP model have less training error and high testing error in comparison with RBFN due its generalization capability. Finally, although, we have conducted several experiments, the ANFIS have the high training and test error with respect to other learning algorithms. Comparing their performance using the execution time of each method, we have found that the RBFN takes the minimum time and the ANFIS takes the maximum execution time.

Table 2- the comparison of execution time among different connectionist models for training/test data set

	Execution Time in Seconds
MLP	2.14063
RBFN	0.89062
Elman	2.26563
ANFIS	12.7344

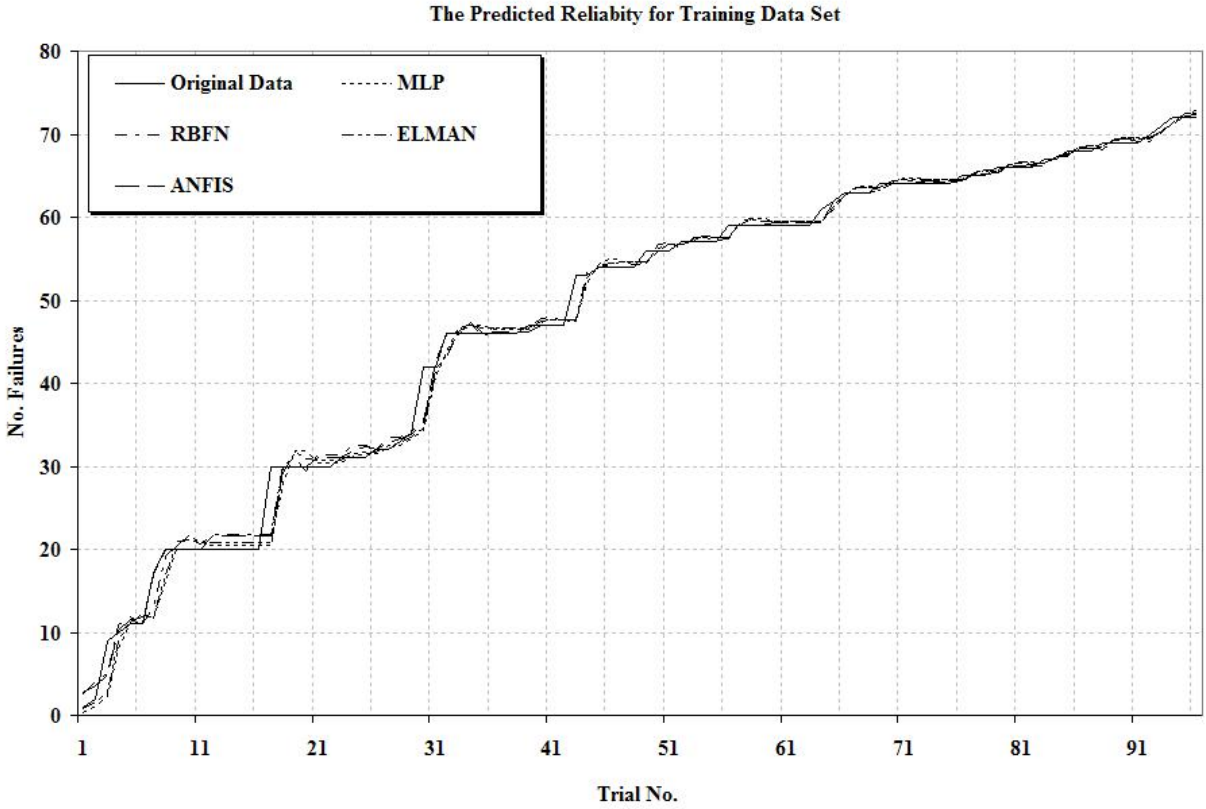

Figure 5: Actual and estimated faults for real time and control application for different models on training data set

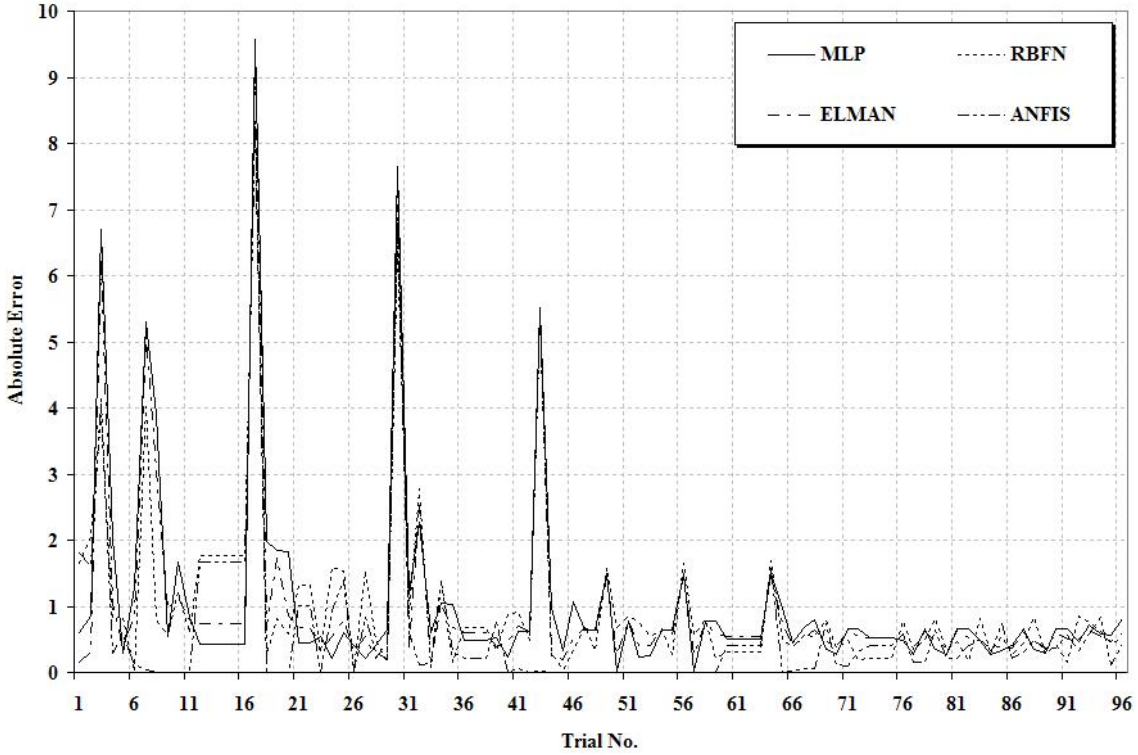

Figure 6: Predicted absolute error for real time and control application training set.

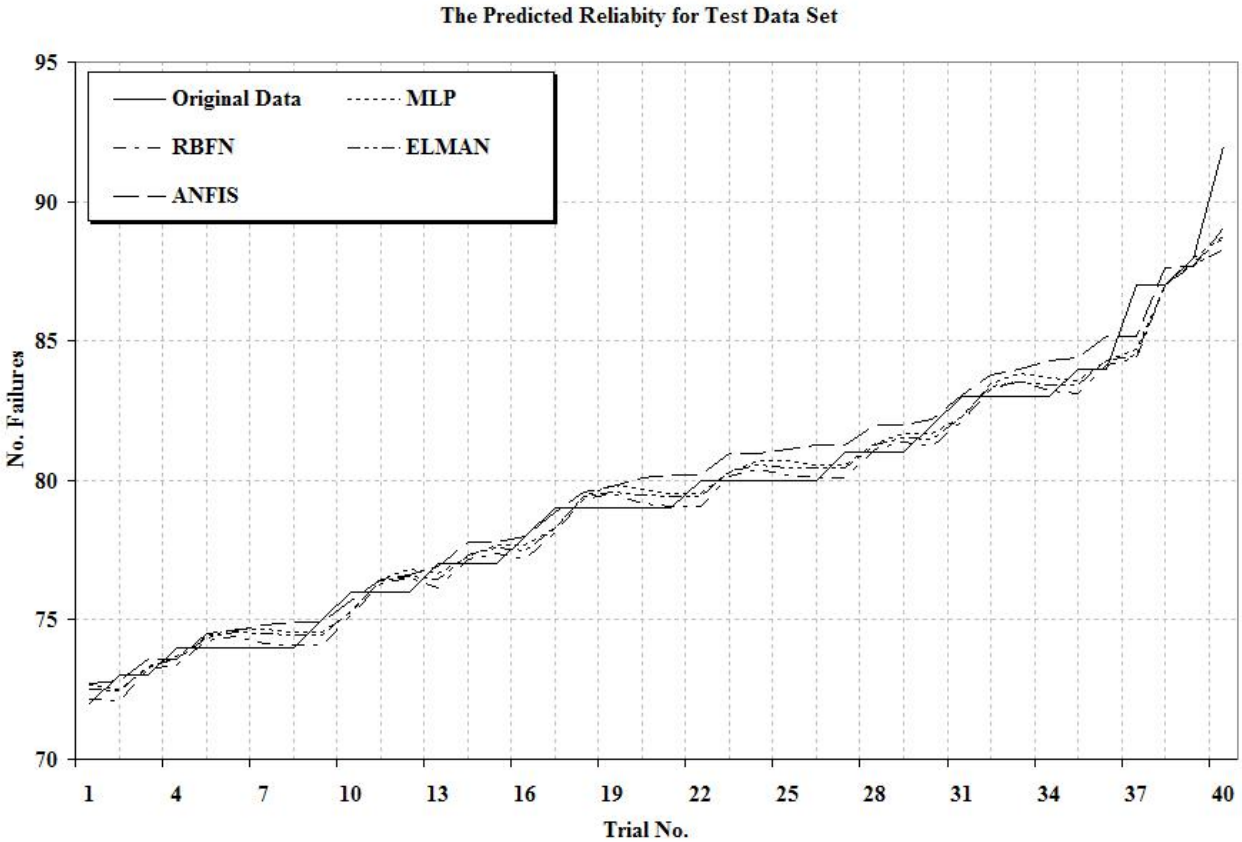

Figure 7: Actual and estimated faults for real time and control application for different models on testing data set

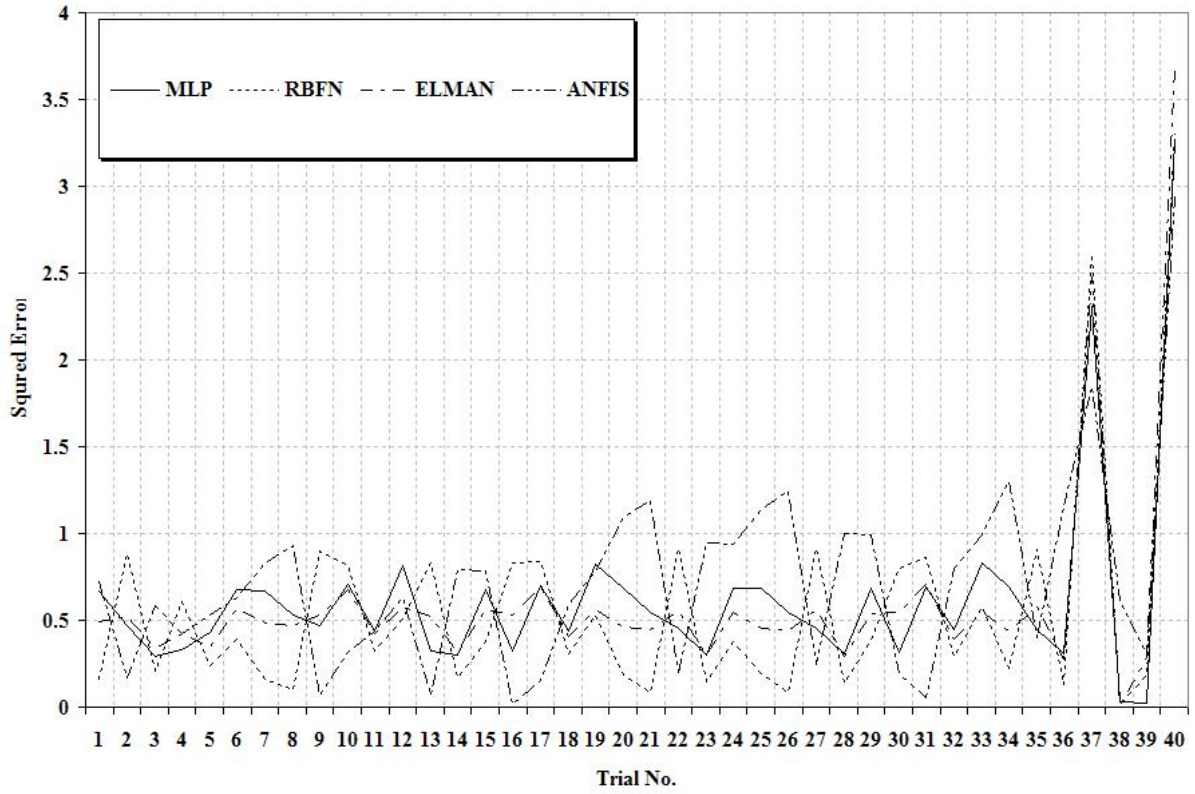

Figure 8: Predicted absolute error for real time and control application testing data set.

6. Conclusions and Future Work

In this paper, we demonstrate the results of utilizing the connectionist modeling and learning algorithms for prediction the software reliability. The demonstration includes a comparison of the four connectionist models with different leaning algorithms or structures. The Elman recurrent neural networks is a robust technique for function prediction due capturing the dynamic behavior of the data set. The preliminary computational results in the MATLAB environment seem quite promising and give insight into the generalization capability of these models.
In the future work, we can explore other soft computing techniques and other different data set.

Acknowledgement

The authors would like to thanks Taif University and King Faisal University for their support. Moreover they are grateful for Dr. M. El-Telbany from Electronics Research Institute, Egypt for reviewing the paper and his valuable comments.

Bibliography

[1] Aljahdali, S. "Prediction of Software Reliability Using Neural Network and Fuzzy logic", Ph.D. Dissertation presented to the faculty of College of Graduate Studies., Dept. of the Software Engineering and Info. System, George Mason University, Fairfax, Virginia, U.S.A, May 2003.

[2] Aljahdali S., and El-Telbany M., "Genetic Algorithms for Optimizing Ensemble of Models in Software Reliability Prediction", In the International Journal on *Artificial Intelligence and Machine Learning* (AIML), V8, ICGST, 2008.

[3] Aljahdali, S., Sheta, A., and Habib, M. "Software Reliability Analysis Using Parametric and Non-Parametric Methods", Proceedings of the ISCA 18th International Conference on Computers and their Application, March 26-28, 2003, pp. 63-66.

[4] Aljahdali, S., Sheta, A., and Rine, D., "Predicting Accumulated Faults in Software Using Radial Basis Function Network", Proceedings of the ISCA 17th International Conference on Computers and their Application, 4-6, April 2002, pp. 26-29.

[5] Aljahdali, S., Sheta, A., and Rine, D., "Prediction of Software Reliability: A Comparison between regression and neural network non-parametric Models", Proceeding of the IEEE/ACS Conference, pp.470-471, 2000.

[6] Demuth H., and Beale M., *MATLAB, Neural Network Toolbox*, version 4.0, 2004.

[7] Ganesan K., Khoshgoftaar T.M., and Allen E.B," Case based Software quality prediction", International Journal of Software Engineering and Knowledge Engineering, 9(6), 1999.

[8] Haykin S., *Neural networks: A Comprehensive Foundation*, Prentice Hall, 1999.

[9] Hu Q., Xie M., and Ng S., *Software Reliability Predictions using Artificial Neural Networks*, Computational Intelligence in Reliability Engineering (SCI) 40, 197–222, 2007.

[10] Jang, J-S.R and Mizutani, C-T., *Neuro-Fuzzy and Soft Computing: A Computational Approach to*

Learning and Machine Intelligence. Eaglewood Cliffs, NJ: Prentice Hall, 1997.

[11] Jang, J-S.R. and Sun, C-T. "Neuro-fuzzy Modelling and Control" *Proceedings of IEEE*, 83 (3) (1995) 378-406.

[12] Karunanithi N., Whitley D., and Malaiya Y., Prediction of software reliability using connectionist models. IEEE Transactions on Software Engineering 18(7) pp. 563-574, 1992.

[13] Khoshgoftaar T.M., Pandya A.S., and Lanning, D.L., "Application of Neural Networks for Predicting Faults", Annals of Software Engineering, 1:141-154, 1995.

[14] Khoshgoftaar T.M., and Allen, E.B., "Logistic Regression Modeling of Software Quality", International Journal of Reliability, Quality and Safety Engineering, 6(4), 1999.

[15] Khoshgoftaar T.M., Allen E.B, Jones W.D., and Hudepohl, J.P., "Which Software modules have faults that will be discovered by Customers?" Journal of Software Maintenance: Research and Practice, 11(1):1-18, 1996.

[16] Liang T., Afzel, N. "Evolutionary neural network modeling for software cumulative failure time prediction", Journal of Reliability Engineering and System Safety, vol.87, pp. 45–51, 2005.

[17] Liang T., Afzel, N. "On-line prediction of software reliability using an evolutionary" connectionist model", Journal of Systems and Software, vol. 77, pp. 173–180, 2005.

[18] Musa, D. J. "Software Reliability Engineering: More Reliable Software Faster and Cheaper" Author house, Indiana, 2004.

[19] Srinivasan, K., and Fisher, D., "Machine learning approaches to estimating software development effort", IEEE Trans, Software Engineering, pp.126-137, 1995.

[20] Stewart, W., "Collinearity and least squares regression", Statistical Science, pp. 68-100, 1987.

[21] Tian, J, "Better Reliability Assessment and Prediction through Data Clustering", IEEE Transactions on Software Engineering, 28(10), 2002.

[22] Tian, J, "Software Quality Engineering", John Wiley and Sons Inc. 2005.

[23] Werbos, P.J. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", *PhD Thesis.* Harvard University, 1974.

Sultan Hamadi Aljahdali, Ph.D. received the B.S from Winona State University, Winona, Minnesota in 1992, and M.S. with honor from Minnesota State University, Mankato, Minnesota, 1996, and Ph.D. Information Technology from George Mason University, Fairfax, Virginia, U.S.A, 2003. He is an associate dean of the college of computers and information systems at Taif University. His research interest includes software testing, developing software reliability models, soft computing for software engineering, computer security, reverse engineering, and medical imaging, also he is a member of ACM, IEEE, and ISCA.

Khalid A. Buragga is an Assistant Professor of Information Systems department in the college of Computer Sciences and Information Technology at King Faisal University, Hofuf, Saudi Arabia. He received his B.Sc. in Computer Information Systems from King Faisal University. And, he received his M.Sc. in Computer Information Systems from University of Miami, USA, and a Ph.D. in Information Technology from George Mason University, USA. His research interests include Software Design, Software Development, Software Quality, Software Reliability, E-Commerce and Web development, Business Process Re-engineering, and Integrating Systems.