ISSN:0975-9646

Forecasting the Reliability of Software via Neural Networks

Yogita Kansal[1], Shailee Choudhary [2]

[1]*Department of Computer Science,* [2]*Department of Information Technology*
Manav Rachna College of Engineering, Faridabad, India (Delhi)

***Abstract*—Today's era highly depends on the applications built via software and if the quality of software lacks it results in incalculable problems. Hence, the software reliability which is a part of software quality becomes essential to manage. For judging the reliability of software many software reliability models are came into practise. Software reliability engineering declares that any of the model usage is not accurate for prediction in all cases. Furthermore, the neural networks are known for their good prediction. For this reason, the proposed work merges the software reliability model and the neural network model. The work concludes three models comprising the capability to judge the accuracy of the number of failures prediction.**

***Keywords*—Software reliability model, Logistic growth curve model, Goel-Okumuto NHPP model, Yamada S-shaped model, Neural network, Execution time, Cumulative failures, Training, Prediction.**

I. Introduction

In terms of software, the reliability refers to the probability of producing error free output for a specified period of time under a certain environment. Software Reliability model provides us a means to measure the reliability of software. The models applicable to the assessment of software reliability are called SRGM. As the reliability of any software is directly proportional to the number of failures i.e. if a user is rarely observing the number of failures then we can say the software is more reliable. Failure is an event which protects the program from running successfully due to some uncertain problems in the syntax of the code or in the environment provided to the software. In software reliability engineering there are thousands of models for depicting the software reliability but still none of the model is eligible to work with every circumstances. Each model has its own assumptions and its own value on which it can be operated. Some examples of such models are Jelinski-Moranda model, Logistic Growth curve model, Goel-Okumuto NHPP model, Yamada S-shaped model and much more. But in this research work we are focusing on just Logistic Growth curve model, Goel- Okumuto NHPP model and Yamada S-shaped model. The paper also focuses on the mapping of above mentioned models with neural networks. However, the main target of the work is to provide the faithful representation of the reliability prediction.

II. Software Reliability Model

Software reliability model is directly proportional to the number of cumulative failures and the execution time. Cumulative failures are the total number of failures found while executing the program in specified period of time. Proliferations of software reliability models have emerged as people try to understand the characteristics of how and why software fails and try to quantify software reliability [1]. Reliability models or analytical models can broadly be classified into two categories: static models and dynamic models.

Static Model: A static model uses some attributes of the project or program modules to estimate the number of defects in the software. A static model of software quality estimation has the following general form:

$$y=f(x1,x2,\ldots\ldots,xk)+e$$

where the dependent variable y is the defect rate or the number of defects, and the independent variable xi are the attributes of the product, the project, or the process through which the product is developed.. The error term is e. some examples are Halstead's software metric, Mc Cabe's cyclomatic complexity metric [2].

Dynamic Model: Based on statistical distributions, the current development defect patterns are used to estimate end product reliability. It tracks the failure data produced by software system to develop a reliable operational profile of the system over a specified time. Some examples of this model are Error seeding model, Time domain Model, Input Domain model, Fault count model [2].

A software reliability model is known as one of the fundamental technologies for quantitative software reliability assessment, and playing an important role in software project management for producing a highly-reliable software system [3]. SRGM is a mathematical model, which describes the reliability of the software as the defects are identified and repaired. For knowing the best model above the variety of models we need to define some parameters through which the optimal one is carried out. The parameters by which the prediction values have been evaluated are average error, relative error, normalised average error etc. The models that are needed to be compared are:

A. Logistic-Growth Curve Model

In general, as the software defects detected and removed with time, the reliability of the software is increasing i.e. the software reliability growths with the fixed defects. Therefore, under some conditions, the models developed to

predict economic population growth could also be applied to predict software reliability growth. These models simply fit the cumulative number of detected faults at a given time with a function of known form. Logistic growth curve model is one of them and it has an S-shaped curve. Its mean value function and intensity function are [3].

$$m(t)= \frac{a}{1 + k*\exp[-bt]} \qquad a>0,b>0, k>0 \quad (1)$$

$$\lambda(t)= \frac{ab \exp[-bt]}{(1 + k*\exp[-bt])^2} \qquad a>0, b>0, k>0 \quad (2)$$

where a is the expected total number of faults to be eventually detected and k and b are parameters which can be estimated by fitting the failure data.

B. Goel-Okumuto model

The Goel Okumuto model was first proposed by Goel and Okumuto that lies in the category of Non homogeneous poisson process model i.e. NHPP which signifies that the mean value function is non linear. Hence, this model is also known as exponential NHPP model. From the broader perspective the goel-okumuto model is the failure count model by which the software reliability is calculated through calculating the number of faults in the specific time interval. The model further comprises of some assumptions like the cumulative number of failures by time t follows a poisson process and must be independent; defects must be repaired immediately after the discovery and the repair must be perfect. Its mean vale function and intensity function are [3].

$$M(t) = a(1-\exp[-bt]) \qquad a>0, b>0 \quad (3)$$

$$\lambda(t) = ab*\exp(-bt) \qquad a>0,b>0 \quad (4)$$

where a is the expected total number of faults to be eventually detected and b represents the fault detection rate. In fact, it follows that [4] .

$$\lim_{t \to \infty} m(t) = a$$

C. Yamada S-shaped Model

The S-shaped reliability growth model was proposed by Ohba and is the illustrative of the gamma distribution class. Here the per fault distribution is gamma. The software error detection process can be described as an S-shaped growth curve to reflect the initial learning curve at the beginning, as team members become familiar with the software, followed by growth and then leveling off as the residual faults become more difficult to uncover. Its mean value function and intensity function are:

$$m(t)= \frac{a * (1-\exp[-bt])}{1 + k*\exp[-bt]} \qquad a>0,b>0, k>0 \quad (5)$$

$$\lambda(t)= \frac{ab \exp[-bt] (1+kt)}{(1 + k*\exp[-bt])^2} \qquad a>0, b>0, k>0 \quad (6)$$

III. OVERVIEW OF NEURAL NETWORK

Artificial neural networks are a computational metaphor inspired by studies of the brain and nervous systems in biological organisms [5]. Neural networks are likened to non parametric models in the statistical literature. It communicates through the connections between processing elements called neurons, fig 1 shows a neuron. Knowledge is encoded into the network through the strength of the connections between different neurons, called weights, w which can be modified so as to model synaptic learning. The unit computes some function f of the weighted sum of its inputs.

While designing the neural network the individual element inputs are $x_1, x_2, \ldots\ldots, x_R$ are multiplied by the weights $w_{1,1}, w_{1,2}, \ldots\ldots, w_{1,R}$ and the weighted values are fed to the summing junction. Their sum is simply Wx, the dot product of the single row matrix W and the vector x. R is the number of elements in the input vector.

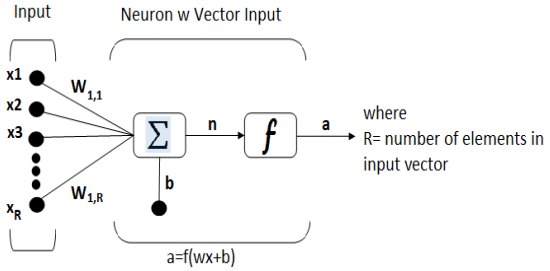

Figure1: A simple neuron

$$a= f(\sum w_{1,R}\, x_R + b) \qquad (7)$$

- The weighted sum is $\sum w_{1,R}\, x_R$ called the **net input** to unit 1, often written net_1.
- Note that $w_{1,R}$ refers to the weight from unit R to unit 1 (not the other way around).
- The function f is the unit's **activation function**. In the simplest case, f is the sigmoid function, and the unit's output is

$$F(n) = 1/(1+e^{-n}) \qquad (8)$$

Neural networks learn by example. The *learning rule* is provided with a set of examples (the *training set*) of proper network behavior $\{x_1,t_1\},\{x_2,t_2\}, \ldots\ldots, \{x_Q,t_Q\}$ where x_q is an input to the network, and t_q is the corresponding correct (*target*) output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. The *Perceptron learning* rule falls in this supervised learning category. There is also the *Supervised Hebbian learning.*

IV. MAPPING NEURAL NETWORK WITH SOFTWARE RELIABILITY MODELS

As mention in section 2 we have multiple software reliability models, among these models we have selected three models i.e. logistic growth curve model, Yamada s-shaped model and Goel okumuto model. Instead of taking a single model we have taken multiple so that we can show the comparison of these models and carried out the best one. The mean value function for these models is as shown in equation (1), (3), (5).

A. *Designing Activation function for the proposed reliability model*

How the software reliability models are merged with neural network has been shown now on wards. As for prediction through neural network, one needs to train the network through input and target pairs. And also for training, the activation function has an important role. That is why here we have decided to map reliability model with neural network by designing the new activation function such that the mean value function of the reliability model is treated as the transfer function or the activation function. The new activation function for each of the model is as depicted in equation (9), (10), (11)

$$F(n)= \frac{1}{1+\exp[-w_{1,R}\, t_{1,R}]} \quad \text{logistic growth curve (9)}$$

$$F(n)= (1-\exp[-w_{1,R}\, t_{1,R}]) \quad \text{Goel-okumuto NHPP model (10)}$$

$$F(n)=(1-((1+ w_{1,R}\, t_{1,R})*\exp[-w_{1,R}\, t_{1,R}])) \quad \text{Yamada S-shaped model (11)}$$

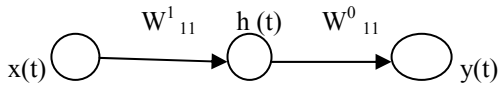

Figure 2: Feed forward neural network

Where x(t) is the input i.e. the time at which a fault is detected in the software, w(t) are the weights that is being randomly selected within the range of -0.5 and 0.5, h(t) is the output of the hidden unit which is gained after applying the activation function as mentioned in equation (9), (10), (11) and y(t) is the desired output i.e. the predicted faults.

B. *Working of Proposed model*

As discussed in previous section, a neural network relies on the output calculated through the activation function. If we combine the software reliability models like GO model, the Delay S-shaped and the logistic growth curve model with neural network, then we need to use $1\text{-}e^{-x}$, $1\text{-}(1+x)\, e^{-x}$ and $1/1+ e^{-x}$ as the activation functions in the hidden layer.

We can map software reliability prediction problem in terms of a neural network through:

$$P: \{(I_k(t), O_k(t)), i_{k+h}(t+\Delta)\} \rightarrow o_{k+h}(t+\Delta) \quad (12)$$

where, $(I_k(t), O_k(t))$ represents the failure history of the software system at time t used for training the network and $o_{k+h}(t+\Delta)$ is the network's prediction. Firstly, our task is to successfully train the network. After that the network is thoroughly used to predict the total number of faults to be detected at the end of future session k + h by feeding $i_{k+h}(t+\Delta)$ as its input.

C. *Prophecy Approach*

We have chosen the testing and debugging data from an actual project described by Yoshira Tohma say Data Set 1(taken from [6] Table 4) for generating an accurate prediction. In the data set, execution time was reported in terms of days and faults in terms of cumulative faults at the end of each day. The total testing and debugging time was 46 days and there were 266 faults.

The process of training a neural network involves tuning the values of the weights and biases of the network to optimize network performance as defined by the network performance function.

Initially, we have trained our network with 100 epochs and 5 hidden neurons by updating the weights at each time we have encountered the wrong output. The calculated gradient error has been plotted as shown in fig 3, fig 4, fig 5.

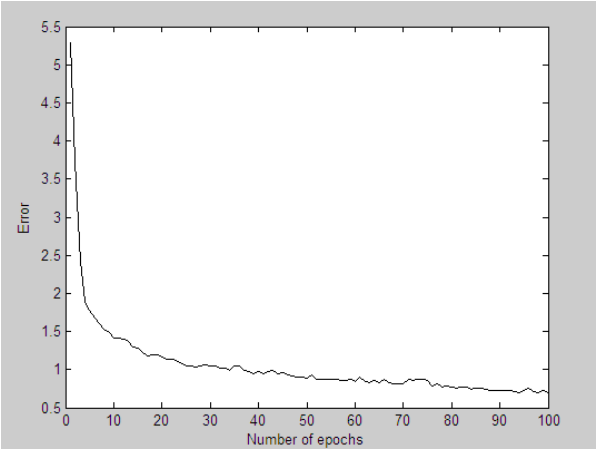

Figure 3: Plot of errors w.r.t epochs of logistic growth curve model

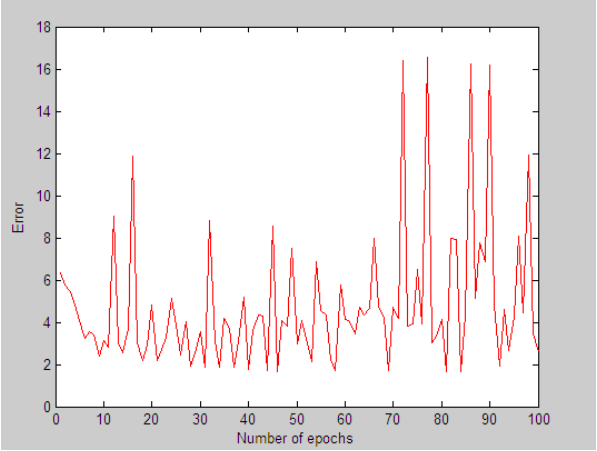

Figure 4: Plot of errors w.r.t epochs of GO NHPP model

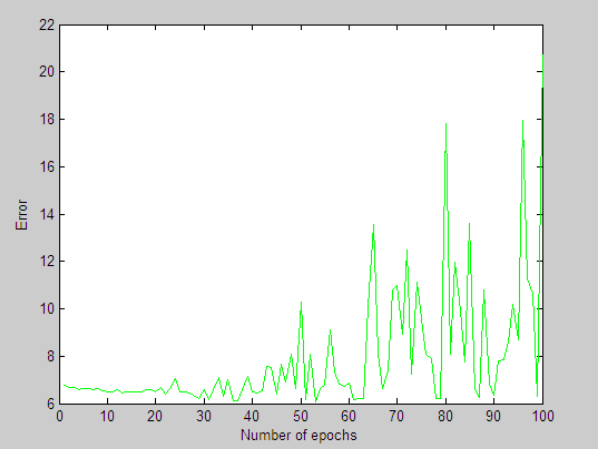

Figure 5: Plot of errors w.r.t epochs of Yamada S-shaped model

D. Comparison Criteria

In order to compare the predictive accuracy of different models, we have used some quantitative measures. The software reliability research community uses a variety of metrics for comparison of models. Let the data be grouped into n points (t_i ,f_i), i=1 to n, where f_i is the cumulative number of faults found at time t_i and t_i is the accumulated execution time in disclosing f_i faults. Feed the pairs of { t_i ,f_i } to train the network by the back-propagation algorithm. t_i (where i>n) acts as a input to the network. Let μ_i be the projected number of faults at time t_i which will be the output of the network after the successful training. A two component predictability measure consisting of average error (AE) and average bias (AB) was used by Malaiya et al. to compare predictive accuracy of the models.

The comparison criteria we engage to compare various model's performance are described as follows [7].

1. The Relative Error (RE) [7]

$$RE = \frac{\mu_i - f_i}{f_i} \qquad (13)$$

where f_i is the actual faults and μ_i is the predicted faults at the end of testing.

2. The Average Error (AE)
3.

$$AE = \frac{1}{n-1} \sum_{i=1}^{n-1} |RE| \qquad (14)$$

the AE measures how well a model predicts throughout the test phase. The average error is calculated for overall testing and debugging session.

4. The Average Bias (AB)

$$AB = \frac{1}{n-1} \sum_{i=1}^{n-1} RE \qquad (15)$$

All the parameters mentioned above can be used to compare the predictive accuracy of models within a single data set only.

V. PREDICTION RESULTS

At this point, we have trained our network through different activation functions. Moreover, fig 3, fig 4 and fig 5 depict the flow of gradient error with the three models. After training the networks, the next step is to find the simulated results of the inputs by which we can calculate our first comparison parameter i.e. relative error. The figure 6 represents the relative error with respect to the execution time for the three models respectively.

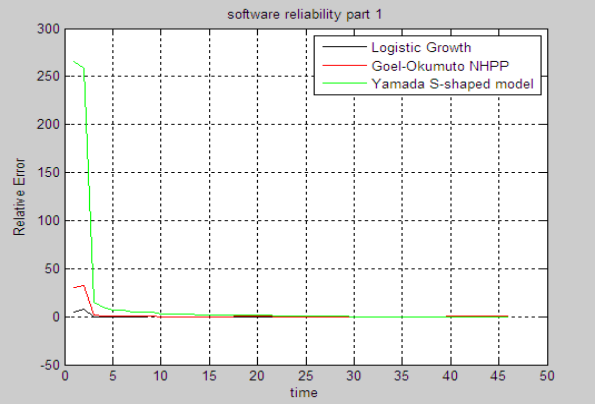

Figure 6: Plot of relative errors w.r.t time of the three models

Further, the average error and average bias of the models are as shown in Table 1:

Table 1: Shows the values for three models

Model	Average Error	Average Bias
Logistic Growth Curve Model	0.3190	0.2672
Goel-Okumuto NHPP model	1.5470	1.5444
Yamada S-shaped model	13.5657	13.2864

VI. CONCLUSION

While dealing with logistic growth curve model, Goel-okumuto NHPP model and Yamada s-shaped model, we have seen that there is only minor variance among these models. As the lower values of average error and average bias can result in more reliable software. We conclude that Logistic growth curve model is the best approach that one can use for the software reliability prediction. Further we can say, logistic growth curve> Goel-okumuto NHPP model> Yamada s-shaped model.

REFERENCES

[1] Pan Jiantao, Spring 1999, "Software Reliability", Carnegie Mellon University

[2] Mariam Rahmani, Azad Azadmanesh, 2011, "Exploitation of Quantative Approaches to Software Reliability", Technical Report, University of Nebraska at Omaha.

[3] Gargoor AI. G Rita, Saleem N Nada, July 2013, "Software Reliability Prediction using Artificial Techniques", Mosul University, Iraq

[4] Aggarwal Gaurav, Gupta V.K, Jan 2014, "Software Reliability Growth Model", University of Rajasthan.

[5] R. Lippmann, , Apr. 1987, "An Introduction to Computing with Neural Nets," IEEE Acoustics, Speech and Signal Processing, pp-4-22.

[6] Y. Tohma et al, 1990, "Parameter Estimation of the Hyper-Geometric Distribution Model for Real Test/Debug Data," ,Tokyo Institute of Technology.

[7] Lyu, M.R; 1996, "Handbook of Software Reliability Engineering", McGraw-Hill.