Time Series Prediction of Debian Bug Data Using Non-Linear Auto-Regressive Neural Network

Jayadeep Pati[1]
1:Corrensponding Author

Department of Computer Engineering
Indian Institute of Technology(BHU)
Varanasi, India
e-mail: jayadeeppati@gmail.com

K.K. Shukla
Department of Computer Engineering
Indian Institute of Technology(BHU)
Varanasi, India
e-mail: kkshukla.cse@iitbhu.ac.in

Abstract— **It is very important for both software project managers and software end-users to monitor and predict the increasing or decreasing trend of bug number in a software system. Software managers can make timely decisions, such as effort investment and allocation of resources by predicting the bug number of a software system accurately. The objective of this paper is to model the bug number data per month as time series and, and analyzing the time series using Artificial Neural Network(ANN). A Nonlinear autoregressive model(NAR) with embedded delay and feedback loop is used for prediction of time series debian bug data. This paper gives a complete neural network approach to bug number prediction. A comparison of five mostly used neural net Training algorithms is given in this paper. The results shows a substantial improvement in performance of Levenberg–Marquardt algorithm with Bayesian Regularization than other training algorithm. The results are confirmed on bug data extracted from bug Ultimate Debian Database(UDD) which is publicly available.**

Keywords*— *ANN; UDD; Bayesian Regularization; Autoregressive; Bugzilla.

I. INTRODUCTION

Bug Prediction is an challenging aspect in the area of software engineering. The prediction result denotes the number of bugs added for a certain period such as per month in a software system, which can be used as an important measure for software managers and software end-users. It enables them to keep track of software quality, and then help them in planning and allocation of resources to complete the fixing tasks.

Large software projects being developed and maintained for many years usually use bug tracking system such as Bugzilla to manage bugs. Software repository contains rich information about software development process and software characters, which are valuable for researchers to gain further insight into the nature and evolution of software projects [1]. The focus of this project is on monitoring and predicting the increased number of bugs per month in a large software system using historical bug data. the various table text styles are provided.

Time series prediction refers to the process of predicting a future measure or measures by analyzing the trend of past and current ones[2]. We used time series analysis to monitor and predict bug number and analyze the data from bug tracking system of Debian over the past 12 years. Bug numbers were calculated per month, and all these bug numbers form a time series. We applied ANN to predict Debian bug number added per month. We show the capability of Neural Net as a better predictor of non-linear time series data. We have also compared five most advanced training algorithm for time series prediction of Bug data. They are:

- Steepest descent algorithm.
- Steepest descent with Momentum.
- Resilient Back propagation (RProp).
- Levenberg–Marquardt algorithm[8,9].

- Levenberg–Marquardt algorithm with Bayesian Regularization.

The proposed methods in this project, which are based on time series analysis, have been applied to Debian bug data. Ultimate Debian Database (UDD) gathers a lot of data about various aspects of Debian in the same SQL database. It allows users to easily access and combine all these data[udd]. There are different types(12) of bugs available and we have considered all the bug types. Debian is a desktop as well as server operating system.

Debian GNU/Linux using the Linux kernel and GNU OS tools is a popular and influential Linux distribution [3]. Debian project, which was started in August 1993, has been developed continually for over 15 years. We collected the Debian bug number data from January 2000 to Feb 2013, that is, 156 months in sum.

The rest of the paper is organized as follows: Section 2 presents a description Time Series Analysis. Section 3 describes about ANN & it‟s role in time series prediction. Section 4 describes about experimental design. Section 5 describes on Results and interpretation. Section 6 concludes the paper & show direction for future work.

II. TIME SERIES ANALYSIS

Definition of Time Series: An chronological sequence of data values of a variable at equally spaced time intervals.
Applications: The usage of time series models is twofold

[5]:

• To have an understanding of the underlying forces and structure that produced the observed data.

• To construct a model and proceed to forecasting, monitoring or even feedback and feed forward control.
Time series analysis accounts for the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for.

Techniques: The construction of time series models can be an ambitious undertaking. There are many methods of model fitting including the following[13]:
- Box-Jenkins ARIMA models
- Box-Jenkins Multivariate Models.

More formally, a time series {xt} can be defined as

a function x of an independent variable t, generating from a process for which a mathematical description is unknown. The main characteristic of a time series is that its future cannot be predicted exactly, as can that of a known deterministic function of t. However, a time series behavior can sometimes be anticipated by describing it through probabilistic laws. Commonly, time series prediction problems are approached either from a stochastic perspective[13] or, more recently, from a neural network perspective[10]. Both approaches has advantages and disadvantages: stochastic methods are though fast, but has limited applicability since they commonly employ only linear models. The NN methods, insted, are powerful enough, but selecting an appropriate architecture, learning algorithm & parameters is crucial task. Theoretical work shows that NNs are effective in uniform approximation of almost any arbitrary continuous function on a compact domain[7]. In addition to their ability of representing complex nonlinear functions, NNs also effectively construct approximations for unknown functions by learning from examples (known outcomes of the function). This ability of approximating unknown complex input-output mappings makes them attractive in practical applications[7].

Chaotic Time Series: A chaotic process is a process where positive feedback of some kind is always involved. Under some circumstances such processes can create time series that shows completely random behavior. Chaotic systems are never fully predictable; because of the feedback simulation and the real series will always show rapid divergence. Chaotic time series commonly occur in sciences, engineering and finance studies.

Predicting chaotic time series:

Suppose, We are given a finite portion of a time series p(t), where p is a component of a vector x that represent: a variable evolving according to some unknown dynamical system. We assume t hat the trajectory x(t) lies on a manifold with fractal dimension D . Our goal is to be able to predict the future behaviol without knowledge of the other components of the vector x(t). In fact, Takens embedding theorem [12] ensures that, under certain conditions, for almost all τ and for some m <= 2D+ 1 there is a smooth smooth map Rm ->R such that: The value of m used is called the embedding dimension and the smallesl value for which below is true is called the minimum embedding dimension, m*. (τ : Time Delay).

$x(n\tau)=f(x((n-1)\tau, x((n-2)\tau, \ldots, x((n-m)\tau))$……….(1)

Therefore, for a known „f" we can predict x at time nτ uniquely by m values in the past.

The Mackey Glass time Series:

The Mackey-Glass equation is the nonlinear time delay differential equation[14]:

$dx/dt=\beta x\tau /1+x\ \tau\ n-\Upsilon x, \Upsilon, \beta, n>0$…………….(2)

where β , γ , τ , n are real numbers, and xτ represents the value of the variable x at time (t−τ). This equation displays a range of periodic and chaotic dynamics depending on the values of the parameters. Here, the of this equation came from a biological perspective. Here we have predicted our time series bug Data using this equation.

III. ANN IN TIME SERIES PREDICTION

Artificial Neural Networks (ANNs) are relatively complex electronic models similar to the neural structure of the brain. ANN transforms inputs into outputs using best of its ability.

Neural network architecture for time series analysis:

The following Neural Nes are mostly used for time series analysis.
Elman Neural Network:

Define abbreviations an Jeffrey L. Elman in 1990 developed a simple recurrent neural network (SRN) called Elman neural network. This network has an input layer, a hidden layer, and an ouput layer. ElmanNet also has a context layer. There is a connection without weight from the output of the hidden layer and the context layer . The Elman network stores the value and outputs them on the next run of the neural network. These values are then sent, using connection having weights, back into the hidden layer. Elman neural networks are very useful for predicting sequences, as they possess a limited short-term memory.

Jordan Neural Network:

The Jordan Neural Network[15] is a simple recurrent network (SRN) developed by Michael I. Jordan in 1986. There is a connection from the previous output from the output layer to the context layer which echos that value back to the hidden layer's input. The previous iteration's output layer provides input to the hidden layer.. Jordan neural networks are generally trained using GA, simulated annealing, or one of the propagation techniques. Jordan neural networks are commonly used for prediction.

Time delay neural network (TDNN):

Time delay neural network (TDNN) is an alternative neural network architecture whose primary purpose is to work on real-time/future data. The advantage of this architecture is to adapt the network online and hence helpful in many real time applications, like time series prediction.

The architecture has a continuous input that is delayed and sent as an input to the neural network. As an example, consider training a feed forward neural network being trained for a time series prediction. The desired output of the network is the present state of the time series and inputs to the neural network are the delayed time series (future values). Hence, the output of the neural network is the predicted next value in the time series which is computed as the function of the past values of the time series.

A dynamic network, which consists of a feedforward network with a tapped delay line at the input is called the focused time-delay neural network (FTDNN).

NARNET:
NAR (nonlinear autoregressive) neural networks can be trained to predict a time series from that series past values.

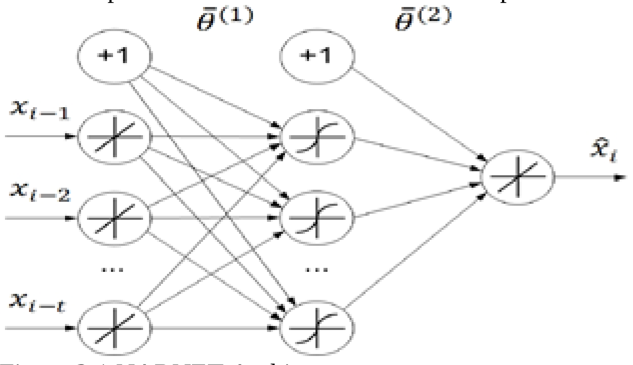


Figure 2.1 NARNET Architecture.

Here, we have used a nonlinear autoregressive neural network as a prediction model of Debian bug nuber information. This is a complete neural network approach to bug number prediction.

A. Training of ANN for Time series analysis:

Once a network has been structured for a particular application, it is ready for training. At the beginning, the initial weights are chosen randomly and then the training or learning begins. There are two approaches to training; supervised and unsupervised. In supervised training, both the inputs and the targets are present. The network then processes the inputs and matches its resulting outputs with the desired outputs. Now the Errors are back propagated through the system, causing the system to adjust the weights, which control the network. In unsupervised learning method, there is no target output for the network. It is as if there is no teacher to present the desired patterns and hence system learns by discovering and adapting to structural feature in the input patterns.

LEARNING LAWS (ALGORITHMS):
The problem of neural network learning can be seen as a function optimization problem, where we are trying to determine the best network parameters (weights and biases) in order to minimize network error.

The steepest descent algorithm, also known as the error backpropagation (EBP) algorithm dispersed the dark clouds on the field of artificial neural networks and could be regarded as one of the most significant breakthroughs for training neural networks. The EBP algorithm is still widely used today; however, it is also known as an inefficient algorithm because of its slow convergence. We will discuss about the mostly used learning methods in the next section.

Learning Algorithm Types:
In this paper we have done a comparison of five most advanced learning algorithms. They are :

1. Steepest descent algorithm.

2. Steepest descent with Momentum.

3. Resilient Backpropagation (Rprop).

4. Levenberg–Marquardt algorithm.

5. Levenberg–Marquardt algorithm with Bayesian Regularisation.

1. Steepest descent algorithm:
The steepest descent algorithm is a first-order algorithm. It uses the first-order partial derivative of total error function to find the minima in error space. Commonly, gradient g is defined as the first-order derivative of total error function(E). The error function is the SSE(Sum Squared Error). Now the equation of gradient is:

$$g = \frac{\partial E(x,w)}{\partial w} = \left[\frac{\partial E}{\partial w_1} \quad \frac{\partial E}{\partial w_2} \quad \cdots \quad \frac{\partial E}{\partial w_N} \right]^T$$

…………..(1)

With the definition of gradient g in eq^n.1, the update rule of the steepest descent algorithm could be:

$W_{k+1}=W_k-\alpha g_k$………………(2)

where α is the learning constant (step size).

2: Steepest descent with Momentum:
There is another way to improve the the rate of convergence of by adding some inertial momentum to the gradient expression. This can be accomplished by adding a portion of previous weight change to the current weight change. A commonly used rule for updating is given by Rumelhaet et al.(1986) which includes a addition of momentum term. The updation equation is given by:

$W_{k+1}=W_k-(\alpha g_k+\mu[\Delta w_k])$…………(3)

Where μ is the momentum co-efficient. The value of μ should be less than one. The typical values lies between 0.5 to 0.9.

3. Resilient backpropagation:
Martin Riedmiller and Heinrich Braun in 1992 gave the Rprop, short for resilient backpropagation, as a learning heuristic for supervised learning in feedforward artificial neural networks. This is a first-order optimization algorithm. Rprop considers only the sign of the partial derivative over all patterns (not the magnitude), and operates independently on each "weight". For

each weight, if there was a change of sign of the partial derivative of the total error function compared to the last iteration, the update value for that weight is multiplied by a factor η−, where η− < 1. If in the last iteration there is no change in sign, the update value is multiplied by a factor of η+, where η+ > 1. The update values are calculated for each weight as described above, and finally each weight is changed by its own update value, in the opposite direction of that weight's partial derivative, so as to minimise the total error function. η+ is empirically set to 1.2 and η− to 0.5. In general backpropagation we have

$\Delta W_{ij}=\alpha\,(\partial E/\partial W_{ij})$ ....(4)

In RProp we have :

In RProp we have :

$\Delta W_{ij}=-\text{sign}\,(\partial E/\partial W_{ij})\,\Delta_{ij}$...............(5)

4: Levenberg–Marquardt algorithm:

The Levenberg–Marquardt algorithm [8,9] blends the steepest descent method and the Gauss–Newton algorithm. Fortunately, it inherits the speed advantage of the Gauss–Newton algorithm and the stability of the steepest descent method. It's more robust than the Gauss–Newton algorithm, because in many cases it can converge well even if the error surface is much more complex than the quadratic situation.

By Newton's Method, the Gradient can be represented in term of hessian matrix i.e.2nd order derivative of Error function :

$-g=H\Delta W$…………(6)

Where H is called the Hessian Matrix which is a second order partial derivative of error function.

Now we can write,

$\Delta W=H^{-1}g$.......................(10)

Therefore, the update rule for Newton's method is:

$W_{k+1=}W_k-H_k^{-1}g_k$................(11)

As the second-order derivatives of total error function, Hessian matrix H expresses the proper evaluation on the change of gradient vector. By comparing with gradient descent one may notice that well matched step sizes are given by the inverted Hessian matrix.

If Newton's method is applied for weight updation, in order to have Hessian matrix H, the second-order derivatives of total error function have to be calculated and it could be very complicated.

In order to simplify the calculating process, Jacobian matrix J(First order partial derivative) is representation given by Gauss–Newton Algorithm. Afer derivation the update rule of the Gauss–Newton algorithm is presented as:

$W_{k+1=}W_k-(J_k^TJ_k)^{-1}J_ke_k$................(12)

Obviously, the advantage of the Gauss–Newton algorithm over the standard Newton's method is that the former does not require the calculation of second-order derivatives of the total error function, by introduction of Jacobian matrix J instead.

In order to make sure that the approximated Hessian matrix J^TJ is invertible, Levenberg–Marquardt algorithm introduces another approximation to Hessian matrix:

$H=J^TJ+\mu I$......................................(13)

where

μ is always positive, called combination coefficient

I is the identity matrix.

Hence, the update rule of Levenberg–Marquardt algorithm can be presented as:

$W_{k+1=}W_k-(J_k^TJ_k+\mu I)^{-1}J_ke_k$................(14)

As the combination of the steepest descent algorithm and the Gauss–Newton algorithm, the Levenberg– Marquardt algorithm switches between the two algorithms during the training process. When the combination coefficient μ is very small (nearly zero), and Gauss–Newton algorithm is used. The cases where the combination coefficient μ is very large, the steepest descent method is used. Here by comparing with steppest descent we have,

$\alpha = 1/\mu$…………………..(15)

Limitations (Levenberg–Marquardt algorithm):The Levenberg-Marquardt is very sensitive to the initial network weighs. Also, it ignores the outliers in the data, which causes overfitting noise.

5. Levenberg–Marquardt algorithm with Bayesian Regularisation:

MacKay (1992) has proposed a Bayesian framework which can be directly applied to the neural network learning problem. Bayesian regularization introduces a new term 'cost function' which search not only for the minimal error, but for the minimal error which intern uses the minimal weights. It works by introduction of two Bayesian hyper parameters, alpha and beta, to tell which the direction (minimal error or minimal weights)of the learning process. The cost function will then become:

- $C(k) = \beta^*E_d+\alpha^*E_w$, where:

1. E_d is the sum of squared errors, and
2. E_w is the sum of squared weights

By using Bayesian regularization, one can avoid the costly cross validation. Regularization also eliminates the rigorous testing of some hidden neurons for a problem. A third variable, gamma, denotes the number of effective weights which are used by the network, thus giving an indication on how complexity of the network. Many practical realizations of Bayesian Regularization generally do the updating of the hyper parameters alpha and beta after each training cycle. However, according to Poland (2001) the most popular update algorithm fails to produce robust iterates if there is not much training data. The following algorithm shows the updating process of the hyper parameters. Both LM and LM-BR methods base their calculations in the inverse Hessian matrix.

IV: EXPERIMENTAL DESIGN

1: Data Collection

In this research, we study bug number growth per month, and we obtain the original data from the public Debian bug data that is available Ultimate Debian Database. We analysis the Debian bug data from March 2000 to August 2013. The original data in the bug tracking database is like this: “"id";"Last Modified": “#607021”;" 2010-12-14 ". There 11 types of bugs available found in UDD like PHP Bugs, Ruby Bugs, Localisation bugs ,etc. We have considered all types of bugs in our analysis. In order to do time series analysis, we transform the data and get the bug number added per month. Therefore, we get a time series with 164 observations totally. The whole series . From the year 2000 to the year 2013, the bug series has significantly increasing trend. Here, we have performed ADF (augmented Dickey-Fuller Test) is used to test the non-Stationary behavior of Debian bug number series. The behavior of time series is non-stationary. The plot of the a portion series from year 2008-2012 is given below.

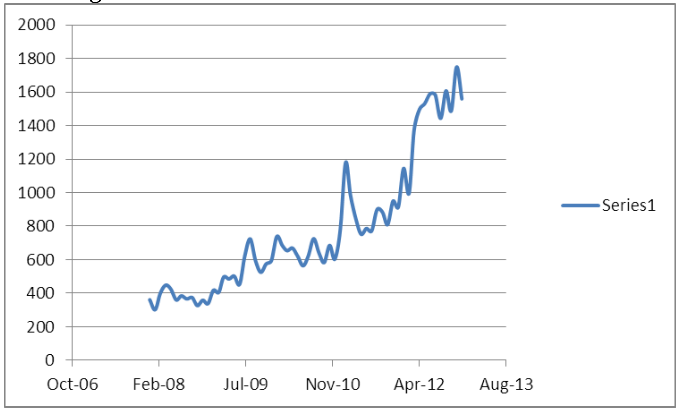


Figure 4.1: Time Series Plot

2. Design of Experiments:

The time series bug data is divided into training set and test set. Inside training set some data are used for validation testing. The entire data is divided into four experiments. For 1st experiment bug data from 01-01-2000 to 01-12-2009 is used for training and validation and 01-01-2010 to 01-06-2010 is used for testing. Similarly, the process is repeated for adding more data to the training set and testing it for next six months. Here, we are performing 6-step ahead prediction for the bug series. The experiments are ordered as given in table 1.

3. Data Normalisation:

The data need to be normalized before being trained by the neural net. Normalisation is very important process before implementation of algorithms in matlab environment. ALL DATA must be normalized, i.e. all values of attributes in the dataset has to be changed to contain values in the interval [0,1]. The input data normalization with certain criteria, prior to training process, is crucial to obtain good results, as well as to fasten significantly the calculations[16]. The data are Normalized using the formulae:

New_data= Old_Data/Maximum(All Data).

Table1: Data Partion for Experiments

Experiment Number	Training Data	Test Data/Forecast Data
1	01-01-2000 to 01-12-2009(152 data values)	01-01-2010 to 01-06-2010(6 data values)
2	01-01-2000 to 01-12-2010(144 data values)	01-01-2011 to 01-06-2011(6 data values)
3	01-01-2000 to 01-12-2011(132 data values)	01-01-2012 to 01-06-2012(6 data values)
4	01-01-2000 to 01-08-2012(126 data values)	01-09-2012 to 01-02-2013(6 data values)

4. Implementation of Algorithms:

Here we have a taken a Nonlinear autoregressive neural network for predicting the time series bug data. The Network has some predefined input delay. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors. The linear output layer is most often used for function fitting (or nonlinear regression) problems. Here we have taken 2 hidden layer with 6 and 3 neurons in 1st and 2nd layer respectively. In our network the relationship between the output Y_{t+1} and the inputs $Y_{t,}$ Y_{t-1}, Y_{t-2}, Y_{t-3} has the following representation:

y(t+1) = y(t) - b*y(t) + c*y(t- τ)/(1+y(t- τ).^10);

where:

b = 0.1;
c = 0.2;
τ (Time delay) = 17;

We have trained the data using four algorithm((1) steepest descent algorithm, (2) Steepest descent with Momentum (3) Resilient Backpropagation (Rprop) (3) Levenberg–Marquardt algorithm. (4) Levenberg–Marquardt algorithm with Bayesian Regularisation. Here we are predicting six steps ahead i.e. prediction of Bug Data for next six months(as in Table 2). We have Trained our time series data and tested them upon test dataset given in the table 2. We have calculated the MSE for each of them. (MSE: Mean squared normalized error performance function).

V: RESULTS OF EXPERIMENTS

1: Code Execution and MSE calculation:
All the experiments are implemented in Matlab environment. For Each experiment the data are trained and validated for the predefined training set and tested for the predefined test data. We have plotted graphs comparing comparing the validation, training, prediction and error data for all the five algorithm below.

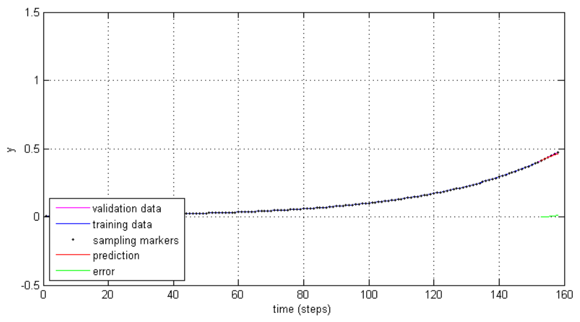


Figure 5.1:Plot for Levenberg–Marquardt algorithm

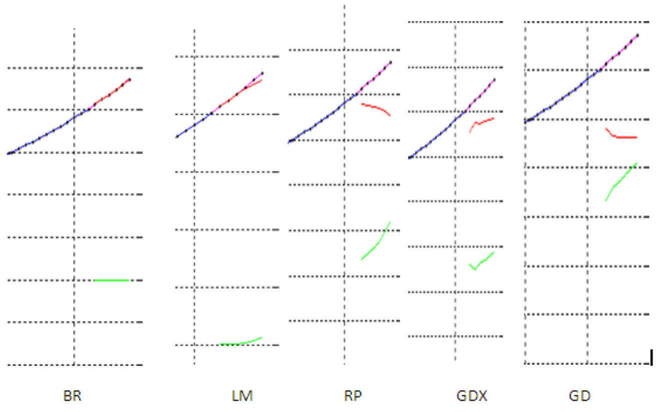
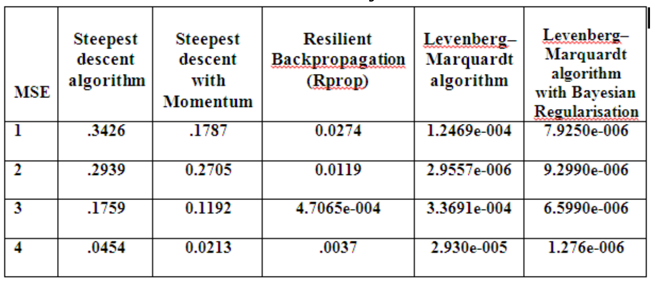


Figure 5.2: Zoomed View of plot for 5 algorithms
From Left 1:LM-BR, 2: LM, 3: RProp, 4: Steepest Descent, 5: Steepest With Momentum

We have calculated the MSE(Mean Squared Error) value for each which is tabulariesed as given in table 2:

Table 2: Calculated MSE Values by NARNET



MSE	Steepest descent algorithm	Steepest descent with Momentum	Resilient Backpropagation (Rprop)	Levenberg–Marquardt algorithm	Levenberg–Marquardt algorithm with Bayesian Regularisation
1	.3426	.1787	0.0274	1.2469e-004	7.9250e-006
2	.2939	0.2705	0.0119	2.9557e-006	9.2990e-006
3	.1759	0.1192	4.7065e-004	3.3691e-004	6.5990e-006
4	.0454	0.0213	.0037	2.930e-005	1.276e-006

2: Interpretation and Analysis:

To properly interpret our tabulated result we have plot a scatter plot for MSE values calculated by each algorithm in Y- axis and time(year) in X- axis. The graph is given below.

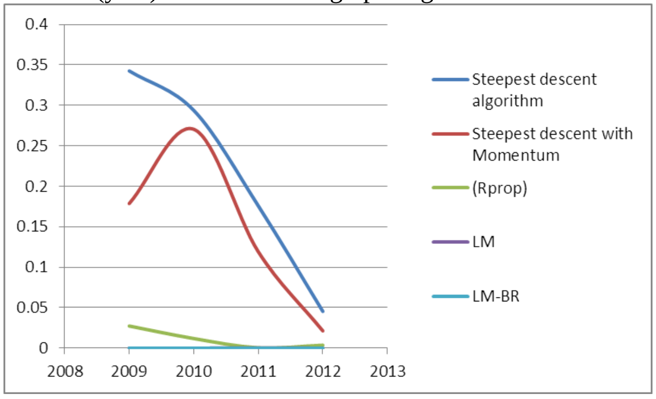


Figure 5.1: MSE Comparison

From the Graph we understand that MSE for Levenberg–Marquardt algorithm with Bayesian Regularisation is lowest than all other algorithm. Also we know that the MSE vlaue decreases as more Training Data is available.

We have done a statistical analysis to check the better performance of the LM-BR over LM. For each of the two algorithm we perform 10 iteration for experiment no 4 as given in table 2 and collected the MSE values in Table 3:

Table 3: MSE for LM and LM-BR

Sl No:	LM Algorithm	LM algorithm BR
1	2.05E-04	4.05E-06
2	8.14E-06	6.46E-06
3	1.31E-04	1.33E-08
4	2.74E-04	8.80E-06
5	2.84E-04	3.50E-06
6	7.64E-04	7.70E-06
7	1.38E-04	2.55E-06
8	7.49E-04	2.64E-09
9	2.85E-04	1.72E-09
10	3.77E-04	1.63E-08
Average	3.31E-06	3.22E-04

We have found the mean MSE for Levenberg–Marquardt algorithm with Bayesian Regularisation is less than Levenberg–Marquardt algorithm. Here, we have plotted a graph comparing the values of MSE of LM & LM-BR below.

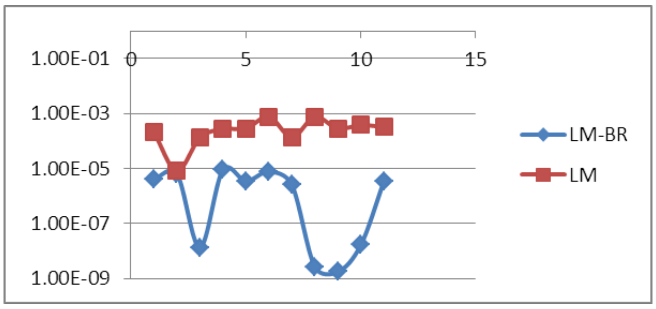


Figure 5.3: Graph showing comparison between MSE values LM & LM-BR

The graph explains that the MSE value by Levenberg–Marquardt algorithm with Bayesian Regularisation is always less than Levenberg–Marquardt algorithm.

I. Conclusion and Future work

This paper adopts time series analysis to monitor and predict the trend of software bug number using a nonlinear auto regressive Neural Net. The paper gives a complete neural net approach to bug number prediction. The paper also Compares five advanced training algorithm for the Nonlinear autoregressive neural network and we have shown that Levenberg–Marquardt algorithm with Bayesian Regularisation is the most advanced algorithm which give the least MSE value.

In the future we will study performance of time series analysis comparatively with other methods like Ensembling different technique like PSO, GA and Neuro-Fuzzy system to have some better output. We will also apply time series analysis to large open source software systems.

References

[1] Hassan, A.E. The road ahead for Mining Software Repositories. Frontiers of Software Maintenance, 2008. FoSM 2008. page(s): 48-57.

[2] Taeho Jo, VTG schemes for using back propagation for multivariate time series prediction, Applied Soft Computing, Volume 13, Issue 5, May 2013, Pages 2692-2702, ISSN 1568-4946, 10.1016/j.asoc.2012.11.018.

[3] **http://udd.debian.org/bugs.cgi**K. Elissa, "Title of paper if known," unpublished.

[4] Box, G. E. P., and Jenkins, G. (1994), Time Series Analysis: Forecasting and Control, Holden-Day. 3rd Edition, Prentice-Hall: New York, NY

[5] Wenjin Wu , Wen Zhang ,Ye Yang ,Qing Wang, "Time series analysis for bug number prediction" , 2nd International Conference on Software Engineering and Data Mining (SEDM), 2010, IEEE Publication Year: 2010 , Page(s): Page(s): 589- 596

[6] R.J.Frank, N.Davey, S.P.Hunt, "Time Series Prediction and Neural Networks" Department of Computer Science, University of Hertfordshire, Hatfield, UK.

[7] Drossu, R. ; Obradovic, Z, "Rapid design of neural networks for time series prediction", Computational Science & Engineering, IEEE Volume:3 Issue: 2, Digital Object Identifier: 10.1109/99.503317 Publication Year: 1996 , Page(s): 78- 89.

[8] K. Levenberg, A method for the solution of certain problems in least squares, Quarterly of Applied Mathematics, 5, 164–168, 1944.

[9] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM Journal on Applied Mathematics, 11(2), 431–441, June 1963.

[10] Connor J, "Recurrent neural networks and time series prediction", IJCNN-91-Seattle International Joint Conference on Neural Networks, 1991., Volume: 1, Page(s): 301 - 306.

[11] Kozma, R., "Time series prediction using chaotic neural networks: case study of IJCNN CATS benchmark test", 2004 IEEE International Joint Conference on Neural Networks, 2004. Proceedings, Volume: 2 , Page(s): 1609- 1613.

[12] F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.S.Young, editors, Dynamzccd Systems and Turbulence. Springer-Verlag, Berlin, 11981.

[13] Mukherjee, S, "Nonlinear prediction of chaotic time series using support vector machines", VII. Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing [1997], Page(s): 511- 520 .

[14] M.C. Mackey and J. Glass. Oscillation and chaos in physiological contol systems Sczence, 197:287, 1977.

[15] Jordan, M.I. (1986). Serial order: A parallel distributed processing approach (Tech. Rep. No. 8604). San Diego: University of California, Institute for Cognitive Science.

[16] J. Sola and J. Sevilla, "IMPORTANCE OF INPUT DATA NORMALIZATION FOR THE APPLICATION OF NEURAL NETWORKS TO COMPLEX INDUSTRIAL PROBLEMS"