ANALYSIS OF FEED FORWARD MULTI LAYER NEURAL NETWORKS FOR SOFTWARE EFFORT AND COST ESTIMATION

A Dissertation

Submitted by

SHEENA THAKUR

11004936

To

Department of Computer Science Engineering & Information Technology

In partial fulfilment of the Requirement for the Award of the Degree of

Master of Technology in

Computer Science

Under the guidance of

Mr. Harshpreet Singh

Asst. Professor

(17478)

Lovely Professional University

Jalandhar

(May 2015)

ABSTRACT

Software effort and cost estimation is one of the critical tasks in the software development process but still their computation technique and its accuracy are challenging. It is an important aspect of the software development estimation which determines the overall time and effort required and development cost of the software project.

Since, the early and widely used techniques are the parametric or algorithmic, implemented using mathematical equations based on the data of already developed projects, like COCOMO, SLIM but they lack in the robustness as well as the effectiveness. Hence this led to the exploration of the non algorithmic models for the accurate software effort and cost computation.

The aim of this study is to propose an alternative approach for software effort and cost estimation using the artificial neural networks. Artificial Neural Networks are particularly useful in modelling complex non linear relationships and provides high degree of accuracy, reliability and flexibility as the software project progresses and the requirements are elaborated. The results can be analysed using different accuracy criterions on the performance plots.

CERTIFICATE

This is to certify that **Sheena Thakur** has completed M.Tech dissertation titled **Analysis of Feed Forward Multilayer Neural Networks for Software Effort and Cost Estimation** under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation has ever been submitted by her for any other degree or diploma.

The dissertation is fit for submission and partial fulfilment of the conditions for the award of M.Tech Computer Science & Engg.

Date:

Signature of Advisor

Harshpreet Singh

Assistant Professor

ACKNOWLEDGEMENT

I am thankful to different local and global peers who have contributed shaping this dissertation work. At the outset, I would like to express deep thanks to **Prof. Harshpreet Singh** for his constant guidance during my dissertation. As my supervisor he has always encouraged me to remain focussed on achieving my goal. His observations and comments helped me establish the overall direction to the research and move forward with investigation in an interesting manner. He has helped me greatly and been a source of immense knowledge.

I would like to thank all my friends and class mates for their encouragement, help and cooperation. This can never be penned in words.

I must acknowledge the academic resources that I have got from Lovely Professional University. I would like to thank the administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, this dissertation work is completed by Almighty God's grace and my family's love, patience and understanding.

Sheena Thakur
11004936

DECLARATION

I hereby declare that the dissertation entitled **"Analysis of Feed Forward Multilayer Neural Networks for Software Effort and Cost Estimation"** submitted for the M.Tech degree is entirely my original work and all ideas and references have been acknowledged. It does not contain any work for the award of any other degree or diploma by me.

The matter presented in this report has not been submitted by me for the award of any other degree elsewhere.

Date: Investigator

Registration No. 11004936

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
INTRODUCTION

This chapter gives a description of Software estimation parameters like software size, effort and cost estimation, software schedule including approaches for their efficient computation.

1.1 Software Estimation

Software estimation is one of the most crucial aspects of the software project development. Software project management and control are not feasible without reliable estimates.

It includes the estimation of software size, cost, effort and schedule involved in the development of the software [Brykczynski et al.,2006]. The precise effort estimation of the software project is the one module that needs more attention.

Software development estimation is classified as:

- Software Size
- Software Effort
- Software Cost
- Software Schedule

1.1.1 Software Size

Software Size is the predictable size or length of the piece of software. Size can be said as the inherent feature of the software piece like weight as the inherent feature of a substantial material. It is a vital characteristic of the software because it affects staffing, budget and duration for the project.

The metrics currently being used for determining the software size are Lines of Code (LOC), Function point (FP) and Object Points [Abran et al.,2009]. Size should be calculated as precise as possible since it is the basis for effort and cost estimation.

i) Lines of Code

Line of code is the most popular software size indicator. It measures the software size simply by counting the number of source instructions in the code of the software project.

It is the program length which effectively predicts the effort as well as the ease of maintenance.

Source Lines of Code

This software metric keeps an account of the total number of lines in the source code of the program [Gencel et al.,2006].

Two types of SLOC are:

a) Physical SLOC: This includes the number of lines in the source code of the program with comments and even blank lines in special cases.

b) Logical SLOC: This includes the number of “executable” lines in the program’s source code. It varies for different programming languages.

For C or C++, logical SLOC can be stated as the number of statements ending with the semi colon.

The models used for SLOC calculation are analogy model, bottom-up model, expert judgement model and parametric model.

Other metrics are:

- KLOC (Thousand Lines of Code) for 1,000 Lines of code: It is the estimated number of deliverable lines of code for project.
- MLOC (Million Lines of Code) for 1,000,000 Lines of Code

Though, LOC can be easily determined after the completion of the project but determining it at the early phase of the project is tough.

ii) Function Points

Function Point facilitates the determination of the size of project independent of the language used in coding. It works by counting the number of externals (number of inputs, number of outputs, number of inquiries, number of files, and number of interfaces) that make up the software system [Symons et al.,1991].

Function Point Analysis

a) Account the occurrence of each external.

b) Assign the complexity weight to each occurrence.

c) Multiply complexity weights and occurrences (function count is obtained).

d) Multiply function count and value adjustment factor to obtain function point count.

$$V=\sum_{i=1}^{14} v_i*0.01+0.65$$

where v_i is rating of 0-5 dependent of some factors affecting software size.

Table 1.1: Factors of function point analysis

	Multipliers		
Externals	Low	Average	High
Input(I)	3	4	6
Output(O)	4	5	7
Inquiry(I)	3	4	6
Number of files(F)	7	10	15
Number of interfaces	5	7	10

ii) Object Points

Object points are developed by Banker et al,1994. They are conformed for object oriented software.

Steps for object point analysis:

a) Count all instances for each object type.

b) Assign a complexity weight to each object.

c) Add complexity weights of all objects to obtain Object-Point (OP).

d) Product Object Point (OP) and Reuse Factor (RF).

e) If the program is not new, calculate New Object Point (NOP).

$$NOP = OP\ (1 - RF)$$

f) Expected productivity cost is calculated using New Object Point, that is,

$$Effort = NOP\ /\ Productivity\ cost$$

iii) Halstead Software Science Model

Halstead estimated the number of errors in the program using the source code length and also the volume metrics using primitive measures [P. G. Hamer,1982]:

The length of the program code can be defined as:

$$N = N1 + N2$$

where, N1= number of operator occurrences,

N2= number of operand occurrences

The program volume corresponds to the amount of required storage space and is defined as:

$$V = N \log(n1+n2)$$

where, n1= number of distinct operators in the program,

n2= number of distinct operands in the program.

This model has got decreasing use and support in the recent years.

1.1.2 Effort and Cost Estimation

A sound project should be well constrained within budget and required effort.

By definition, Effort estimation is the phenomenon of computing the realistic effort required for developing and maintaining the software systems and their duration.

The estimates can be done using the following two approaches [Jørgensen ,2004]:

- **Top-Down Estimate:** It includes algorithmic models and analogies. An algorithmic model may have one or more formulae based on productivity factors or software size.
- **Bottom-Up Estimate:** It includes the disintegration of a large schedule and the overall effort comes out by summing up the computed effort for an individual activity of the component tasks.

Prediction Models

The software effort and cost prediction for a project falls under the following two categories [Ramesh K. et al,2014]:

Algorithmic or Parametric Modelling, and

Non Algorithmic Modelling

- **Algorithmic or Parametric Modelling**

Parametric models are implemented using the mathematical equations based on the data of already developed projects.

They are represented using:

i) Holistic Models

The different holistic models are beneficial for new organizations for which the baseline data from historical projects is unavailable. It does not consider the individual activities regarding the software project for effort prediction.

Table 1.2: Holistic Models for Effort Estimation

Name of the Model	Introduced By	Year
SDM (Software Development Model)	Putnam	1978
SLIM (Software Lifecycle Management)	Putnam	1979
COCOMO (Constructive Cost Model)	Boehm	1981
COPMO (Cooperative Programming Model)	Conte, Dunsmuir, Shen	1986

Of the all holistic models, COCOMO is best known parametric model and is widely used [Boehm, 1981].

ii) Activity-Based Model

These activity based models are dependent of the data from the previous projects for accurate effort computation. They consist of the standard development rates for the organizations.

This can be formulated as:

$$Effort = \sum_{i=0}^{n} \left(\frac{PH}{SLOC}\right)_{i,new} \bullet SLOC_{new} + \sum_{i=0}^{n} \left(\frac{PH}{SLOC}\right)_{i,reused} \bullet SLOC_{reused}$$

where $(PH/SLOC)_{i,j}$ is the labour rate for activity i and class j (j can be new or reused), and $(SLOC)_{j}$ is the estimated size (in SLOC) of code for the class j.

The effort calculated will be in person hour unit.

- **Non Algorithmic Modelling**

The non algorithmic methods are based on soft computing [Masoomi,Z.,2013]. The data from the previous projects might be used and are flexible to the changing circumstances during the software development process. They produce efficient results for effort prediction.

Major categories of non algorithmic models are:

i) Neural Network Based Models

These are the computational models inspired by central nervous system capable of the machine learning and pattern recognition with the functionality similar to that of the human brain [Bawa, A et al.,2012]. Most of the neural network based techniques uses back propagation algorithms.

ii) Evolutionary Computation

Evolutionary Computation is a subfield of artificial intelligence that includes the different optimization problems. It uses iterative progress, such as growth or development in a population. Since the evolution can produce highly optimised process and networks, it is used for the software development estimation.

The techniques used can be:

a) Evolutionary algorithms: They are the heuristic search techniques that imitate the analogy of natural evolution. They implement the principle of survival of the fittest. It is a method for solving the parameter optimization problems [Liu, J. et al.,2005]. The main advantage is that evolutionary strategies can easily control the parameters to self adapt rather than changing their attribute values using other deterministic algorithms. After investigating the evolutionary algorithms, postulates regarding why EAs perform up to the mark are given as:

- Selection process yield good solutions
- Candidate solutions provide independent sampling
- Partial solutions can be modified and integrated via genetic operators

b) Genetic algorithms: They are based on the evolutionary concepts of natural selection. They portray an intelligent exploitation of the mechanism, random search used to minimize optimization difficulties [Choudhary K.,2010]. GAs exploits historical data to direct the search into the space of better performance in between the search region. After examining the inter relationship between parameters of the software models, it is observed that effective software effort is obtained using GA.

c) Swarm intelligence: The particle swarm optimization was introduced by Eberhart and Kennedy in 1999, it is inspired from the imitation of social behaviour. Particle defines a prospective problem solution shifting through n-dimensional region. Each particle shares details about the search space. The speed of the particle gets updated as per their and their neighbours preceding best position. It is the only evolutionary computation techniques that do not follow survival of the fittest.

iii) Case Based Reasoning

It is necessary that effort or cost estimation models should deal with the uncertain and undefined attributes of the software engineering. Case Based Models are specifically useful when it is complex to state concrete rules regarding the problem domain, also it may consider expert advice for complementing the available knowledge [Rashid et al.,2012].

Case Based Approaches takes data from the previously developed projects. The project estimator identifies the completed projects with similar characteristics to the new developed project. The effort of the matching source case is further used for the base of the new project. This is a good approach when you have information about some previous projects but not good enough to draw the generalised conclusions about the vital cost drivers or the productivity rates.

iv) Fuzzy Logic Models

These models are based on fuzzy logic. It is a form of many-valued logic; dealing with reasoning that is approximate rather than fixed and exact. The fuzzy logic variables may have a truth value that ranges in degree between 0 and 1.

Fuzzy logic based models are suitable for approximate and indistinct data. Integrating fuzzy constituent with intermediate COCOMO have improved the accuracy while computation of the software effort [Hamdy et al.,2010]. FL offers several unique features that make it a particularly good choice:

- FL can control nonlinear systems that would be difficult or impossible to model only using equations. This initiates control systems that would be deemed not possible for automation.
- FL is not only constrained to few feedback inputs and one or more control outputs, nor is it necessary to measure rat of change attributes in order for it to be implemented.
- It is inherently robust because it does not require accurate, noise-free inputs and it is programmed to fail safely if a feedback sensor quits or is destroyed. Output control is a smooth function despite a wide range of input variations.

Table 1.3: Advantages and Disadvantages of Existing Techniques [Maitreyee et al.,2013][Khatibi et al.,2011]

Technique	**Type**	**Advantage**	**Disadvantage**
COCOMO Levels	Parametric	Fair results, Commonly used	Huge data is needed, Not adaptable to every project
Expert Judgement	Non Parametric	Quick prognosis, Suitable for some exclusive projects	Completion depends on expertise
Function Point	Parametric	Works with almost all software languages, Better prediction than SLOC	Difficult computerization, Quality results are not achieved
Analogy	Non Parametric	Concepts based on real project experiences, not necessary to have special expert	Require details and information from legacy projects
Top-Down	Non Parametric	Requires least project information, Easy and quick to implement	Less information specific Not much substantial
Bottom-Up	Non Parametric	Highly substantial, More information specific	System level costs may not be considered, Effort and time consuming
Neural Networks	Non Parametric	Homogeneous with variant data, Power reasoning	No specific guidelines for network design, Performance is dependent of training dataset
Fuzzy Approach	Non Parametric	Flexible method, Robust, Can control non-linear systems, Training of data is not necessary	Complex to use, Hard to maintain the degree of relevance or significance
Parkinson	Non Parametric	Corresponds with the experience	Lack realistic estimates
Price to win	Non Parametric	Gets the contract easily	Delay in project delivery, Development team have to overtime for completion

The cost of software projects is deduced by the cost of developing the software. In some projects, other kind of costs may incur referred as overhead costs. The overhead costs can be cost of software and hardware equipments and supplies, company overheads required for office area, administration, etc.[Kemerer et al.,1993].

The cost of software project is simply obtained by multiplying the estimated effort (in person-month or person-hour) with the constant labour or manpower cost.

Cost = Estimated Effort x Manpower Cost

So with this reason, we focus on estimating the precise software development effort.

v) Parkinson’s Method

The Parkinson's principle states that, if the work extends to occupy the available volume, the project cost is predicted with the current resources instead of the objective assessment. For say, if the software project has to be completed in 6 months and only 5 people are capable for work, the calculated effort would be 30 person-months. Inspite of generating fair estimation, this approach is not followed widely since it lacks realistic estimates [B.W. Boehm et al.,1996].

vi) Price to win Method

As per this cost estimation method, the software cost is calculated as the best price to win over the particular project. The cost estimation is based on the customer's allocation rather than the software functionality [Khatibi et al.,2011].

For say, if the software effort estimation for a specific project comes out to be 100 person-months but customer is able to afford only 60 person-months, the estimator would try to fit 60 person-months effort so as to win the project. Resultantly, this approach may cause insignificant delay in project delivery or may lead the development team to overtime for project completion.

In absolute sense, there is only comparison between various approaches but still none model perform best at software development effort as well as cost estimation. All the non algorithmic approaches are nearly competitive. Table1.3 gives an overview of the existing techniques of the software effort and cost estimation. They are widely used for hybrid approaches and analysed to obtain effective software effort computation.

Some of such approaches include:

- Generalized Regression Neural Network Model [Reddy et al.,2010]
- COCOMO II tuned with radial basis function network by Gupta U.
- Radial Basis Function Network with regression and clustering algorithms
- Radial Basis Function Neural Network with Genetic Algorithm [Molani et al.,2014]
- Grey relation analysis (GRA) technique with regression and GRA with fuzzy logic [Nagpal G. et al.,2014]

1.1.3 Software Schedule

The software schedule is estimated with the Gantt charts, Network diagrams, PERT Charts, Critical Path Methods developed using the various tools.

1.2 Introduction to Neural Networks

Artificial neural network is a computational conformity inspired by the framework, processing and learning potentiality of human brain.

Such networks are described using the type and number of the neurons representing the linkage between the individual elements and the learning algorithms applied to data to be used within the network. Each neuron consists of a non linear transfer function which gives an output as a resultant of the number of input values at the neuron unit. A weight is allotted to each input value in the connection. The characteristic feature of such neural network organization is that they can learn from the historical project data and its attributes.

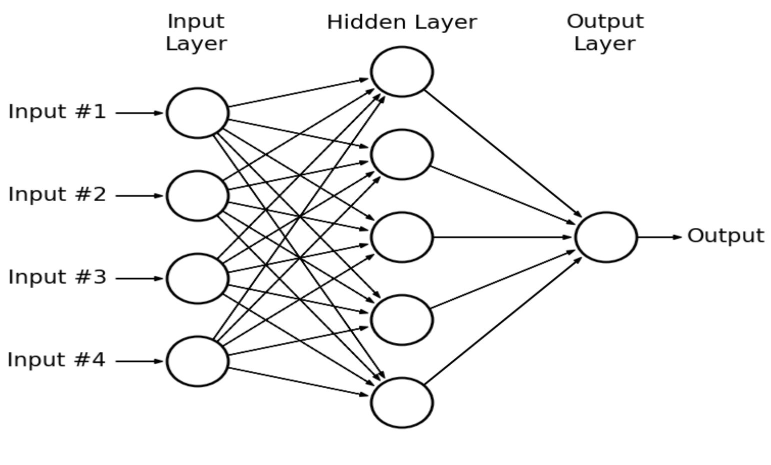


Fig. 1.1: Structure of an Artificial Neural Network

Artificial neural networks are the data modelling tools which are capable of depicting and representing even the complex input/output equations. The main advantage of the neural networks is that they can hold both non linear as well as the linear relationships.
ANNs are preferred because of several characteristics, like they are fault tolerant, generalization and learning capability, tremendous parallelism.

There exist numerous types of neural networks as described in Fig. 1.2. Neural networks majorly used for software effort and cost computation are Cascade Feed Forward, Elman, Recurrent and Feed Forward Artificial Neural Networks.
Other ANNs used for effort and cost calculations are Back Propagation ANN, Wavelet neural networks.

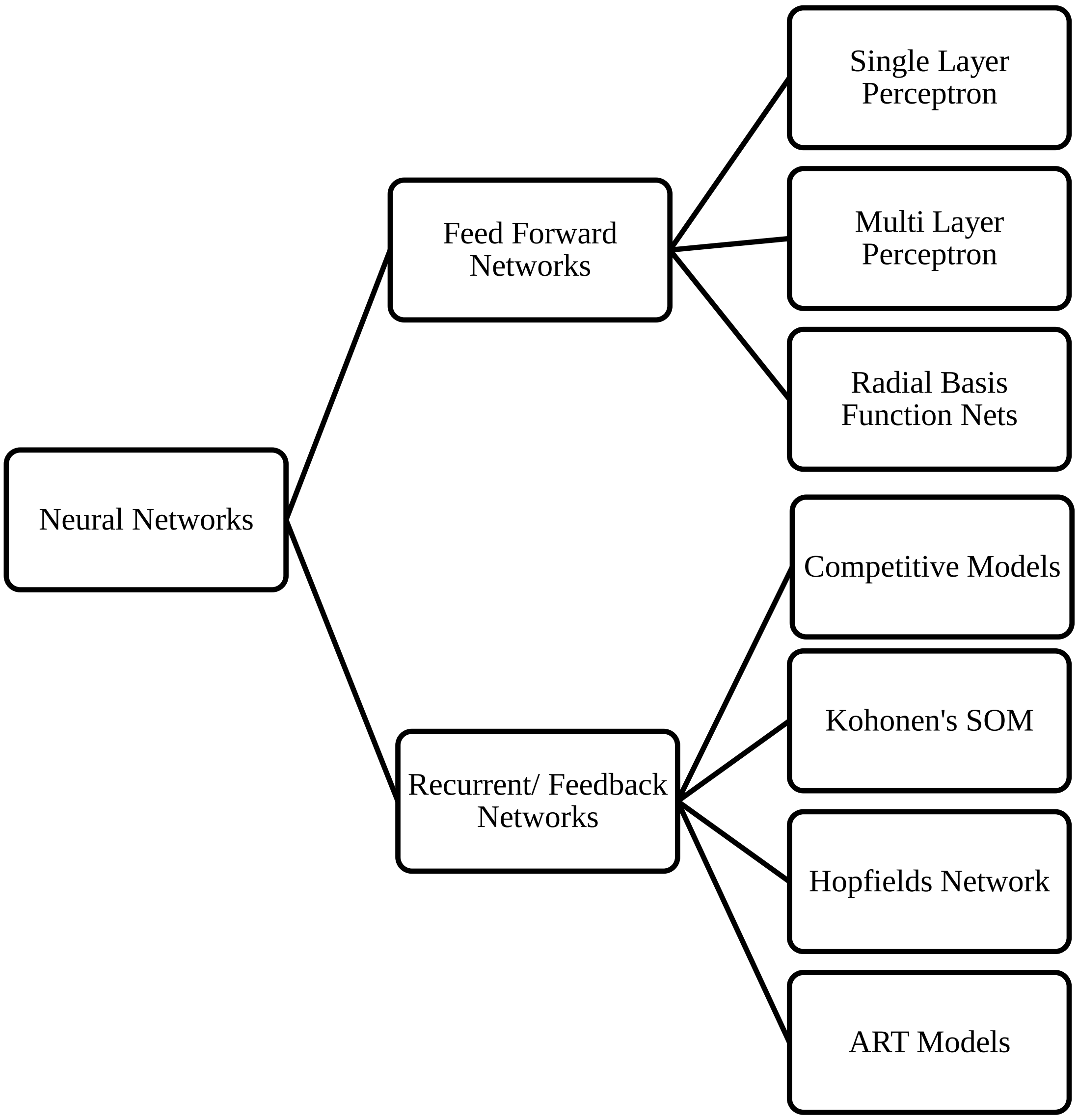


Fig. 1.2: Topology of Neural Networks

A neural network for performing any specific functionality can be trained by adjusting the weights, i.e., values of the connections between individual elements. Generally, the artificial neural networks are trained or adjusted in order to generate input a specific target output from the input given. A neural network can be adjusted on the basis of a comparison of the target element and the output, until the target and the network output resembles each other. Most times such input/target pairs are applicable in the supervised learning for training the neural network. Batch training of such network can be done by altering the weights and bias on the whole batch of input vectors. Incremental training modifies the weights and biases of a network as per the requirement of the different input vectors. Incremental training can also be called adaptive training [Subitsha et al.,2014].

1.2.1 Learning in Neural Networks

The neural network connection weights and biases can be learnt from an arrangement of training representation. Various network structures have learning algorithm approaches.

i) Supervised Learning

The artificial neural network is arranged with a linear output for each input attribute. Weights are predicted to let the network to yield values approximate to the known precise values. For example, back propagation algorithm.

ii) Unsupervised Learning

This type of learning does not need a precise value associated with every individual input attribute in the training dataset. It examines the basic structure of data, relations between the data patterns, and then arranges the patterns into sub categories with the correlations. Kohonen algorithm falls into this category.

iii) Hybrid Learning

It is a blend of supervised learning with the unsupervised. The initial part of weight is calculated via supervised learning, whereas the biases are computed with the help of unsupervised learning.

1.3 Metrics for Assessment of Effort and Cost Estimation

The evaluation for accuracy of the model can be computed using various similarity measures [Prabhakar, M. D, 2013].

Absolute Error is the indicator of the difference between the inferred value x_o and its actual value x:

$$\Delta x = x_0 - x$$

Relative Error is the ratio of mean absolute error to the mean value of the measured quantity:

$$\delta a = \Delta a_{mean}/a_m$$

Table 1.4: Different evaluation measures used for software effort and cost estimation

Evaluation Measure	**Description**	**Formula**
Magnitude of Relative Error (MRE)	Difference between actual and estimated or predicted effort relative to the actual effort	$\frac{\lvert Actual\ Effort_i - Predicted\ Effort_i \rvert}{Actual\ Effort_i}$
Mean Magnitude of Relative Error (MMRE)	Considers the value for each observation in data distribution and is prone to individual prediction with numerous MREs	$\frac{\sum_1^N \mathrm{MRE_i}}{N}$
Median Magnitude of Relative Error (MdMRE)	Median measure is less prone to outliers while calculating the error.	$Median(\mathrm{MRE_i})$
Mean Squared Error (MSE)	It evaluates performance of an estimator/ predictor	$\frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{N}$ y_i= vector representing N predictions, $\hat{y}_i$= vector representing actual values

Root Mean Square Error (RMSE)	It is the root of variance, so this is also called standard deviation.	$\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}$
Mean Absolute Error (MAE)	It is a measure of how far the estimated value lies from the actual values.	$\frac{1}{n}\sum_{i=1}^{n} \lvert P_i - A_i \rvert$ P_i = Predicted Value for point i A_i = Actual Value for point i
Balanced Relative Error(BRE)	This measure gives the ratio of absolute and relative errors.	$\frac{\lvert E - \widehat{E} \rvert}{min\ (E, \widehat{E})}$ E is the estimated effort and $\hat{E}$ is the actual effort
Magnitude of Error Relative (MER)	This error is relative to the estimate.	$\frac{\lvert Actual\ Effort_i - Estimated\ Effort_i \rvert}{Estimated\ Effort_i}$
Mean Magnitude of Error Relative (MMER)	It is mean of MER from N number of observations.	$\frac{\sum_1^N \mathrm{MRE_i}}{N}$
Prediction Level (PRED(p))	Determines per-centage of estimates within a given level of accuracy and is obtained from relative error. Higher the value of predi(p) is better.	$\frac{K}{N}$ N = total size of the data set K = number of programs with MRE less than or equal to the magnitude of p
Correlation Coefficient	Measure of the strength of the relationship between two variables	Higher value of correlation coefficient depicts stronger relationship

1.4 Organization of Thesis

The dissertation report has been organized into following chapters:

Chapter 1 gives introduction about aspects of software development like, software size, effort and cost estimation, and software schedule. A brief description of neural networks is also mentioned following the general metrics used for the performance evaluation.

Chapter 2 outlines the extensive literature review regarding the different non parametric techniques used for the software effort and cost computation.

Chapter 3 discusses the problem formulation, objectives of the study and the research methodology adopted for generating neural network based model for the estimation of software effort and cost.

Chapter 4 highlights the results and discussion regarding various training functions used with the neural networks, their progress, and the performance plots.

Chapter 5 is about conclusion of the research work and the future perspectives in this area.

Chapter 6 is the list of references used in the dissertation completion.

Chapter 2
LITERATURE REVIEW

This chapter highlights the literature review studied during the dissertation.

Prasad Reddy PVGD et al. (2010) proposed neural network models using the Radial Basis and Generalized Regression. COCOMO81 dataset is used for analysis and comparing the results of the proposed model with intermediate COCOMO using different measures MMRE, Mean BRE and Pred(40). GRNN is used for function approximation. It constitutes a radial basis layer and a special linear layer. The first layer corresponds to that of the RBFN but second layer has as many as neurons as input/target vectors. Therefore, it is clearly portrayed that Radial Basis Neural Network gives better results with respect to evaluation measures than Generalised Regression Neural Network and Intermediate COCOMO.

Jagannath Singh et al. (2011) analysed the performance of various Artificial Neural Networks (ANN) in the effort calculation. ANNs can efficiently map functions between dependent variable, effort and independent variable, cost drivers to use it for tool design for software estimation. It generalizes the training data set to predict unseen data. Four types of ANN are pretended for NASA dataset using MATLAB. The four models used are Cascade forward ANN, Elman ANN, Feed forward ANN, Recurrent ANN. The trainlm algorithm is used for training ANN. The actual effort is evaluated with MMRE, RMSE, BRE and Prediction indicator. Cascade feed forward network showed the best among results among the four considered models for cost and effort estimation for software projects.

Ekbal Rashid et al. (2012) proposed the estimation approach based upon the analogy. A case based reasoning model for the calculation of distinct effort for software projects have been developed using Euclidean and Manhattan distance. These distance measures use the knowledge base to figure out the matching cases for the input parameters. Case Based Reasoning (CBR) deals with the uncertainty and imprecision. It may use the expert knowledge as complement to available knowledge. CBR may involve Delphi, group consensus and other expert judgement methods. The CBR model is generally used when it difficult to define precise rules of the problem domain. The CBR is validated on a student dataset and then evaluated using magnitude of MRE. The results are considered quite efficient as development time is a real complex attribute since it is dependent on human behaviour. For this approach, better results are obtained when large database is considered.

Anupama Kaushik et al. (2012) proposed the use of back propagation trained feed forward neural networks for software cost estimation. The proposed model obliges COCOMO II model and enhances the software cost estimation. The model is implemented using the two validation datasets COCOMO and COCOMO NASA 2. Post architecture model of COCOMO II is considered. The back propagation method is used for training the samples and comparing actual values. The evaluation criteria used is Magnitude Relative Error (MRE) and Mean MRE (MMRE). The proposed model definitely outscored COCOMO model with comparable results.

Venus Marza et al. (2012) proposed the neuro-fuzzy model for effort estimation using ANOVA. The expert knowledge, project data and the traditional algorithmic model as well as the fuzzy rules are put together into one general framework. The proposed model is compared with the four other models, viz, neural network model, fuzzy logic model, multiple regression model and statistical model. It is inferred that function point is the only aspect that influences the software development effort; however, the precise estimation should consider the various other elements like type of development, development language, and average number of developers worked on the project in the development environment.

Amanjot Singh Klair et al. (2012) analysed the computer based techniques, Support Vector Machine (SVM) and k-Nearest Neighbour Approach (kNN) for effort estimation for software projects. These techniques facilitate an automatic and economic software tool for generating rank for software by formulating the relationships based on its training. kNN is an instance based learning, the object is classified by majority vote of its neighbours, as object is fixed at the most common class amongst the other k neighbours. Since the survey for various applications have been conducted using SVM and kNN models and it is found that the SVM performs way better than kNN technique with perspective of computation of effort involved in the software program development.

Sriman Srichandan (2012) proposed the construction of Radial Basis Function Networks for the effort and cost estimation of the software. The datasets used for case study are COCOMO81 and Tukutuku. Prediction indicator and MMRE are used for evaluation. . The width of the activation function and the neuron count in the hidden layer of the radial basis structural design influences the cost estimate obtained from them. Different mathematical equations are used for determining the width associated with Gaussian Kernels and it is clear that using unsuitable width would result in poor function approximation. The width should be chosen as per the number of projects covered by centres in region for monitoring the overlap between existing Gaussian kernels for adequate estimates.

Mohd. Sadiq et al. (2013) analysed the organic projects with software size between 2-50 KLOC written in C++ with value of LOC/FP equivalent to 64 for effort calculation. The applied approach is linear regression model predicting software efforts as well as the function point, i.e. Effort= -1.5 + 0.1804 FP. 0.1804 implies that it would cost 0.1804 man day to finish one function point and the software company can establish their own linear model by using their records. The value of function point can be computed as FP = a + b Effort. The relationship between the function point and effort came out to be highly positive correlation, that is, when value of first variable increases or decreases, predicts the same directional change for second one.

Maitreyee Dutta et al. (2013) analysed Arificial Neural Networks (ANN) and Support Vector Machine (SVM) for the China dataset for software effort estimation. Various evaluation measures like MMRE, PRED(25), Correlation coefficient, etc. are being used for the comparison. Error back propagation method is used for the training of ANN. During the forward pass of this propagation algorithm, the weights are assigned in the network and in the backward pass; the weights are rectified and adjusted as per the error calculated. On the other hand, SVM takes a defined set of input attributes, which of two possible classes the input is a member of what makes the support vector machine a non probabilistic binary linear classifier. Resultantly, the model approach with smaller values of MMRE, MAE, RMSE and greater value for correlation coefficient is considered more suitable and efficient.

Usha Gupta et al. (2013) analysed the precise software effort estimation using radial basis function network (RBFN) integrated with the ANN-COCOMO II used for functional approximations. This technique highlights the functionality of COCOMO II with the radial basis clustering algorithms. The dataset used for training the RBFN is COCOMO II. RBFN training is quick to implement when both the stages of RBFN are treated with suitable algorithms. The clustering algorithms used for RBFN are K-means and APC-III. K-means computes the centre of the clusters by minimizing the distance between clusters whereas the APC-III is a one pass algorithm finding the radius of each cluster. Cluster is generated for each dataset. The number of generated clusters will be inversely proportional to the radius of cluster. RBFN is incorporated with the three clustering techniques, COCOMO, K-means (calculating mean value for the Gaussian function of network) and APC-III (estimation based on past projects by calculating radius function). RBFN yields more accurate results with the APC-III.

Ridhika Sharma (2013) reviewed different non algorithmic models and observed that they are definitely more accurate and efficient than algorithmic ones for computing the software effort. Neural network models, fuzzy logic based models, genetic algorithm are discussed in this respect. Neural networks used are the back propagation algorithms and cascade correlation networks. Fuzzy logic models are generally used with vague and uncertain data. The fuzzy triangular membership function and the Fuzzy Inference System (FIS) developed using the COCOMO are explained. The fuzzy based COCOMO model and the COCOMO are compared using various membership functions, and it is observed that the fuzzy based COCOMO is a better choice for effort estimation. Genetic algorithms search heuristic that mimics the process of natural selection. These algorithms are not as efficient as fuzzy and ANN, they offer many limitations too. The outcomes from various research papers and journals proved that if neural techniques are used, MMRE value comes out to be comparatively better. In case of fuzzy approaches, when Gaussian membership function is used, lower MMRE is yielded.

Tulin Ercelebi Ayyilidiz et al. (2014) proposed the early software effort and cost estimation approach using the problem domain concepts inherited from the use cases as developed during the requirement phase. The data of 14 CMMI level 3 certified software development projects using object oriented development approach from defense industry have been used and their effort is computed using the UCP (Use Case Point) Analysis depicting better results with the proposed methodology than the UCP. The results show that the correlation between the number of conceptual classes and number of software classes as well as actual software development effort is at peak. These correlations accounts that the software development size and effort can be dependent of number of concepts for the object oriented software projects.

P. Subitsha et. al. (2014) reviewed five approaches, Multilayered Perceptron Network, Radial Basis Function Network, Extreme Learning Machines, Particle Swarm Optimisation and Support Vector Machines for the grand scale study to obtain unambiguous result for most suitable approach. The data used for the study have been taken from COCOMO II. After investigating about these techniques, accounting their strengths and weaknesses, it is said that the ordinary least squares methodologies are better than the non linear methodologies with respect to the performance. Though the evaluation measures depict minor differences in the absolute terms but they are of important notice in software cost and effort terms. A very simple approach of regression is accounted to be suitable because of incremental stepwise analysis as well as the statistical significance of testing the parameters. Thus it is suggested to concentrate more on the quality rather than clustering various predictive attributes.

Geeta Nagpal et al. (2014) proposed that the analogy is one of the most suited software effort estimation approach when the project is not completely or poorly understood, calculating the effort from the like projects from the project warehouse. They have introduced espousing two approaches, viz., the one is predictive model based on Grey Relational Analysis (GRA) and regression and other deals with the uncertainty in the projects using combination of GRA and fuzzy set theory. GRA is a Grey System Theory technique which uses the point to point distance between cases. In first approach, the effort is calculated using k nearest projects from the sum total of n projects, and then regressing their effort of k projects while in the second approach, the grey relational coefficient makes use of the Fuzzy C-Means (FCM) algorithm to compute the distance between the projects. The research has been done on the six different datasets and it is found that it outperforms the algorithmic techniques. The outcome using both the approaches are precisely treated to statistical testing using the Wilcoxon Signed Rank Test. Resultantly it is proved that the proposed models GREAT_RM and FuzzyGRA can be used at the early phase of the project when data is uncertain and they have shown enhanced effort estimation.

K.P. Manju et al. (2014) proposed a linear regression model facilitating exponential transformation for the estimation of software effort from use case diagrams. The use case diagrams are inputs for software size and use case points are received as corresponding output. This can be efficiently used in initial phases of the software development life cycle for the accurate results from software effort calculation. As the linear regression finds the relationship between variables and for precise results of the regression, the data obtained in the effort should be distributed normally. Exponential transformation of data completes the normal distribution. The exponential function maps the relationship in which an inevitable change in the independent variable also gives the similar proportional change for the dependent variable. Linear Regression applied on the normalized data gives correct results. It is depicted that log linear regression model as well as the linear regression model with exponential transformation are efficient for software effort estimation. The proposed regression based model overcomes the disadvantages and is useful in improving the conformity of software effort calculation.

Jayashree K.M. et al. (2014) proposed a simple analogy based software technique for the effort estimation of software projects. Algorithmic techniques like COCOMO, Use case point method lack flexibility and are not much understandable which gave rise to the introduction of non algorithmic techniques showing the relationship between cost drivers and effort. In such aspect, this paper have focussed on the Case Based Reasoning, also called estimation by analogy. They just gathered the data from past similar projects within a set of attributes namely number of images, team experience, number of developers involved, etc. and analyzed it. It is based on the principle “similar projects require similar effort”. The CBR is a four step process-Retrieve, Reuse, Revise, Retain. The variation in the values of the attributes for similar projects (past project and candidate project) lies within the fixed specified range. The distance with new candidate project is computed by adding up the differences in individual attribute values for every similar project defined. Thus the analogy based technique is quite simple and less time and effort consuming as it determines the new effort value by comparing the already available effort values for the software projects.

Maryam Molani et al. (2014) proposed a methodology for software effort and cost computation using the radial basis function (RBF) neural network and genetic algorithm (GA). They have designed software cost estimation models compatible with the COCOMO81 model using the methodology of RBF neural network and GA enabled the selection of most relevant attributes so as to enhance the training network. The proposed model has RBF trained as per the COCOMO81 dataset and further the reduced model is proposed on the basis of GA. Since training a RBF network is easier than training the multilayered perceptron network, thus involves two step learning process, i.e., input dataset determines the parameters for basis function and in next step, the weights are obtained. Further RBF network is generated using newrb function. GA following survival of the fittest selects the suitable attributes giving high network estimation results. Hence ANNs effectively models complex non linear relationships.

Seyyed Reza Khaze et al. (2014) have proposed the approach of effort estimation using Particle Swarm Optimization and the effective counter measures of PSO and better results have been obtained than the COCOMO Model. In order to obtain good software quality and decreased costs, Software effort estimation is used for the support. The KEMERER dataset is used with the algorithms and evaluation criteria considered is MMRE of the software projects.

Berna Seref et al. (2014) predicted the software effort using Multilayer Perceptron and Adaptive Neuro Fuzzy Inference System. The data sets used are the NASA with 93 projects and Desharnais with data of 77 projects. The results depicted that MMRE of the Adaptive Neuro Fuzzy Inference System is lower than Multilayer Perceptron, where as the evaluation measure PRED(0.25) is higher for Adaptive Neuro Fuzzy Inference System. Hence, Adaptive Neuro Fuzzy Inference System yielded efficient results than the Multilayer Perceptron.

The covered literature reviews numerous techniques for the effort estimation of a software project.

The software estimations are being developed since many years, yet obtaining the exact estimates for the software projects are challenging. It is difficult to compute effort in the initial phase of the software development when software product is unseen. Accurate effort estimation is critical because cost, time and manpower required for the software development also rely on it. The effort estimation techniques are based on algorithmic and non algorithmic modelling. For precise estimations, researchers are working for the development of the new models, improving the existing ones using soft computing or proposing hybrid approaches efficient for the software effort estimation.

The scope of exploring the non algorithmic approaches for prediction of effort arises from limitations of the non-algorithmic approaches. Such limitations can be accounted as:

a) Algorithmic approaches are incapable of handling the practical ambiguous conditions of the software projects. They are not efficient to handle explicit data and are devoid of reasoning capabilities.

b) It is complex to model the inherent associations between subscribing factors using mathematical formulas.

c) The parametric or algorithmic approaches are not as accurate and robust as they should be.

d) Effort and cost computation of the developing projects depends majorly on adapting the model as per the needs of organization, using historical data which might not always be readily available.

e) High error deflection is observed in parametric approaches.

Hence, there is comparison between numerous approaches but still no best technique is evolved for the software effort estimation. Also, there is no universally acceptable approach evolved for accurate effort calculation for all types of software projects, i.e. projects for small scale, medium scale or large scale organizations.

Chapter 3
PRESENTWORK

3.1 Problem Formulation

In earlier researches, various limitations of the parametric approaches of the software effort as well as cost estimation were observed, as mentioned in previous chapter. Thus following problem is formulated to overcome such challenges and limitations.

1. Model the neural network based models using efficient training functions for software effort and cost prediction.

2. Yield the results using performance plots, regression plots, etc.

3.2 Objectives of the Study

In past researches, various limitations were observed, as mentioned earlier. Hence the problem is formulated to overcome such challenges and limitations.

1. Study and Analysis of various evolved approaches for the effort estimation required for the development of software projects.

2. Examine the various Software Effort Estimation Models based on the Artificial Neural Networks to overcome the limitations of algorithmic modelling.

3. To implement the proposed model using MATLAB tool and validate its accuracy using evaluation metric MMRE for the software effort and cost estimation.

3.3 Research Methodology

Research is a logical, methodical and orderly search for new and useful information on a specific subject. It adds contribution to the existing knowledge.
Research Methodology is one of the ways to interpret and resolve a research problem.

In order to achieve the mentioned objectives, the effective methodologies are considered to complete this task as:

1. In order to achieve the objective “Study and Analysis of various evolved approaches for the effort estimation required for the development of software projects”, comprehensive literature survey was carried out for software estimation techniques and different challenges were observed that exist during software development for the effort computation during early design phase.

2. In order to achieve the objective “Examine the various Software Effort Estimation Models based on the Artificial Neural Networks to overcome the limitations of algorithmic modelling”, the work is lead as:
a) First, a complete analysis of the limitations of the algorithmic modelling is done.
b) Then Artificial Neural Network (ANN) based models were accounted potential, accurate and robust for effort computation of software projects.
c) ANNs based approaches were critically reviewed.

3. In order to achieve the objective “To implement the neural network for effort and cost estimation using MATLAB and validate its accuracy using evaluation metrics for the software effort estimation”, the work is encompassed in the following steps:

a) Selection and preparation of dataset: Choose the dataset containing attributes of the software projects required for the feed forward neural networks. The dataset will be dispensed into training set, testing set and validation set. The data for function fit problems will be organized as input matrix, X and target matrix, T.
b) Design and function fitting to the network: Structure the Neural Network, two layered neural network and one hidden layer with activated sigmoid functions. An ANN consists of set adaptive weights in their connections with the nodes. The network is initiated with some weights using randstream along random seed to avoid randomness. The non linear input (cost drivers) is approximated to the output (effort) of the network as the linear combination of all outputs.
c) Training: Train the simple artificial neural network.
The training neural network can be enhanced using variant algorithms, like batch gradient back propagation, resilient back propagation, etc.
d) Testing the neural network: This step gives the sense that how the data will fit to the real world data. Regression for the output and target values will be plotted for all the data attributes for testing the network. In case the values do not fit, more training at the hidden layer is advisable.

e) Error Calculation: Check the performance of proposed model using evaluation measure. MMRE will be calculated as:

i) Compute error.

$$Error = Actual\ Effort - Estimated\ Effort$$

ii) Determine relative error.

$$RE = \frac{Actual\ Effort - Estimated\ Effort}{Estimated\ Effort}$$

iii) Magnitude of relative error will be calculated as:

$$MRE = abs(RE)$$

iv) Calculate Mean magnitude of relative error as:

$$MMRE = \frac{\sum_{1}^{N} MRE_i}{N}$$

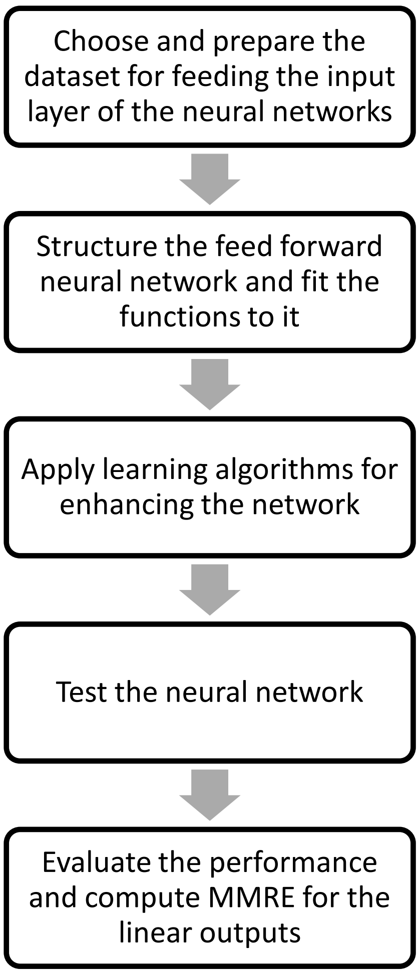

Choose and prepare the dataset for feeding the input layer of the neural networks
Structure the feed forward neural network and fit the functions to it
Apply learning algorithms for enhancing the network
Test the neural network
Evaluate the performance and compute MMRE for the linear outputs

Fig. 3.1: Flowchart of step-by-step process of research methodology adopted

Chapter 4
RESULTS AND DISCUSSION

When the weights and biases of the artificial neural networks are defined, the network can be trained. The multilayer feed forward neural network is trained for function approximation (nonlinear regression) or even pattern recognition as per the application requirement. The training process consists of a set of specified network behaviour—network inputs, `p` and target output, `t`. The phenomenon of training an artificial neural network involves the adapting values of the weights as well as biases of the neural network to elevate the network overall performance, as defined by the function *`net.performFcn`*. The conventional performance function for feed forward neural networks is Mean Squared Error (MSE). It decreases as the network is trained. It is formulated as:

$$F = mse = \frac{1}{N}\sum_{i=1}^{N} {e_i}^2 = \frac{1}{N}\sum_{i=1}^{N}(t_{i-}a_i)^2$$

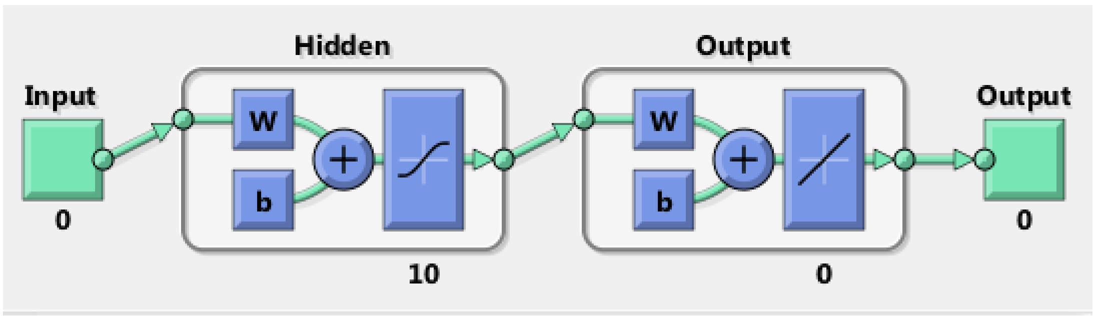


Fig. 4.1: Function fitting neural network (view)

The input and output both have the sizes 0 because the network is not configured to match the input data as well as the target data. This is observed when the network is trained.

Listed are the training functions used with the neural networks from the toolbox of MATLAB R2013a:

i) train: Train neural network

train calls *net.trainFcn(trainlm)*, with the training parameter values indicated by *net.trainParam*. Generally one epoch of training is represented as a single presentation of all input vectors into the network. The network is further upgraded as per the results of all such notations. Training occurs until a maximum number of epochs are achieved, or any stopping condition of the function *net.trainFcn* occurs.This is implemented as:

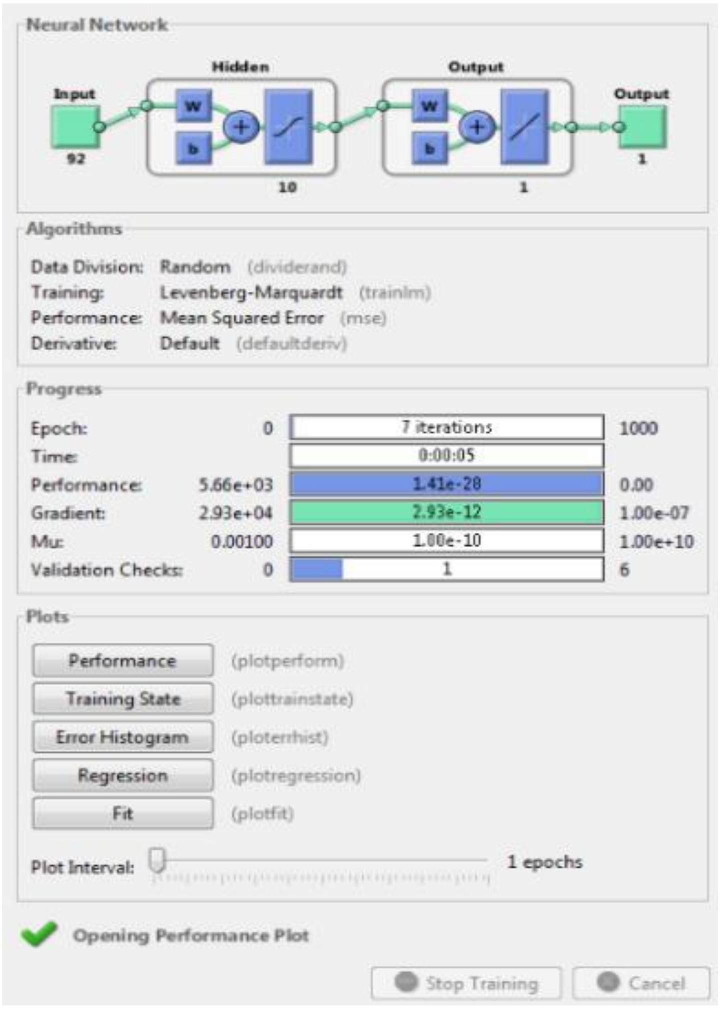
```
% to eliminate initial random values every time we assume some bit
 setdemorandstream(35734567)

% Create Hidden Layer ( 10 Neurons in 1)
net = fitnet(10);
view(net)

% N/w is ready to  trained
[net,tr] = train(net,b1,b);
nntraintool

% Difference b/w actual n expected Price
b1 = double(b1);
y = net(b1);

% Diference calculation
 e = b - y
```



Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	7 iterations	1000
Time:		0:00:05	
Performance:	5.66e+03	1.41e-28	0.00
Gradient:	2.93e+04	2.93e-12	1.00e-07
Mu:	0.00100	1.00e-10	1.00e+10
Validation Checks:	0	1	6

Plots

(plotperform)
(plottrainstate)
(ploterrhist)
(plotregression)
(plotfit)

Plot Interval: 1 epochs

Opening Performance Plot

Fig. 4.2a: Neural Network Training

The data is automatically divided into training, testing and validation dataset. The training will be working until the network is improving over the validation dataset. The NN training tool depicts the network under training as well as the algorithms used for training it. The quickest and default training function is *trainlm* for most feed forward networks. The training state and the conditions that halted it are highlighted in green.

The training window enables to access plots: performance, training state, error histogram, regression and fit.

The **performance plot** depicts values of the performance function versus iteration number. It plots training, testing and validation performances. The **training state plot** depicts proficiency of training parameters, like gradient magnitude, or number of the validation checks, etc. The **error histogram plot** depicts the propagation of the network errors. The **regression plot** depicts the relation of regression between the network outputs and their targets. The histogram as well as regression plots are helpful in validating the network performance.

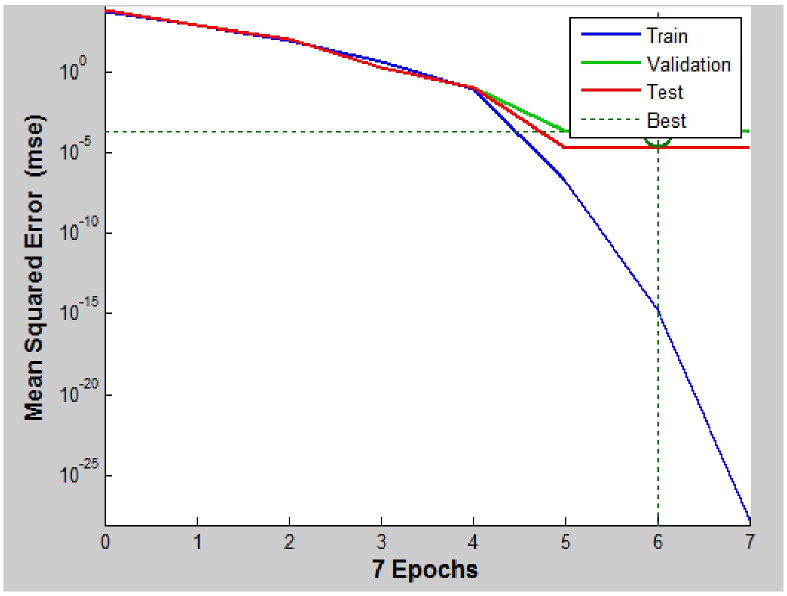


Fig. 4.2b: NN Training Performance (trainlm)

For considering the improved network’s performance due to training, click on “Performance” button on the training tool or PLOTPERFORM can be called.

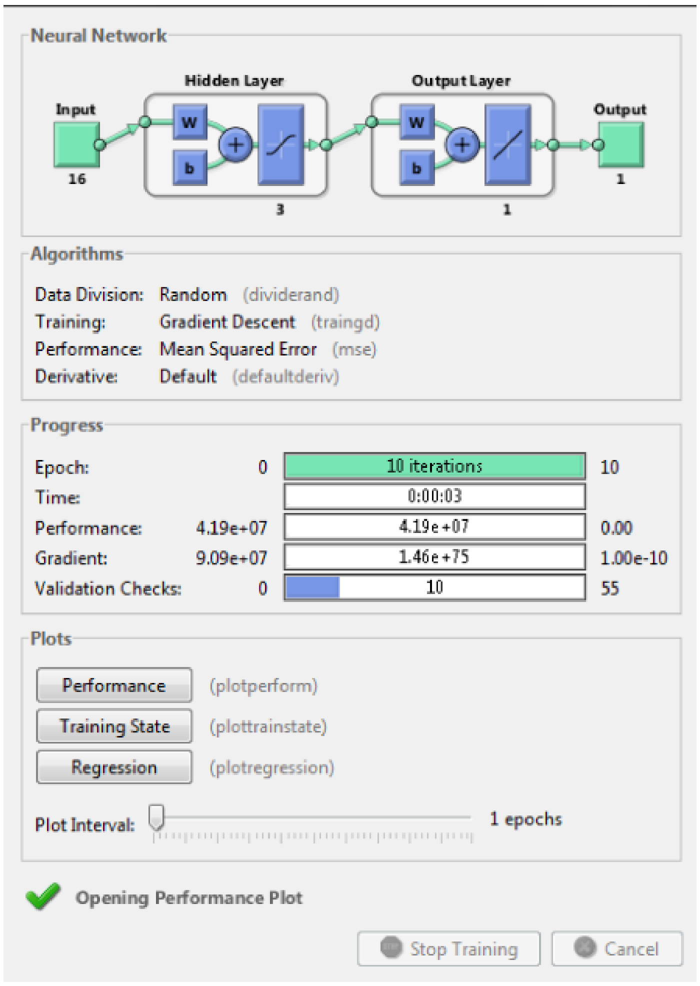
As per the fig. 4.2b, the best validation performance is achieved at 0.00019714 at epoch (repetition) 6.

ii) traingd: Batch gradient back propagation

This is a batch steepest descent training function available in MATLAB. The *traingd* can train any network as long as its weight, input, biases and transfer functions have derivative functions. It is implemented as:

```
net = newff(p,t,3,{},'traingd');

net.trainParam.epochs = 10;     %Maximum number of epochs to train
net.trainParam.goal = 0;        %Performance goal
net.trainParam.showCommandLine  = 0;     %Generate command-line output
net.trainParam.showWindow   = 1;         %Show training GUI
net.trainParam.lr   = 0.05;             %Learning rate
net.trainParam.max_fail = 55;         %Maximum validation failures
net.trainParam.min_grad = 1e-10;      %Minimum performance gradient
net.trainParam.show  = 25;          %Epochs between displays (NaN for no
displays)
net.trainParam.time = inf;             %Maximum time to train in seconds
```



Algorithms

Data Division: Random (dividerand)
Training: Gradient Descent (traingd)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	10 iterations	10
Time:		0:00:03	
Performance:	4.19e+07	4.19e+07	0.00
Gradient:	9.09e+07	1.46e+75	1.00e-10
Validation Checks:	0	10	55

(plotperform)
(plottrainstate)
(plotregression)

Plot Interval: 1 epochs

Opening Performance Plot

Fig. 4.3a: Neural Network Training using traingd

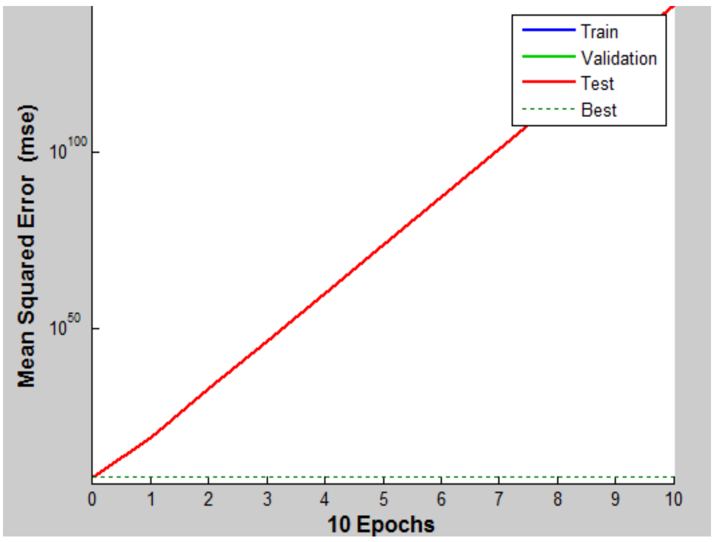


Fig. 4.3b: NN Training Performance (traingd)

Training stops when validation performance has increased more than max_fail times since the last time it decreased (when using validation) and other prevailing conditions.

Performance is quantified in MSE, with log scale as shown in fig. 4.3b. The best validation performance is 3.5284713.2542 at epoch 0. It may reduce as the network gets trained effectively.

iii) traingda: Gradient descent with adaptive learning rate back propagation

Adaptive learning rate is efficient fpr keeping the learning rate stable while keeping the size of the learning step as large as required. The *traingda* can train any network as long as their weight, input, and transfer function does have the derivative functions. Back propagation computes the derivatives of performance *dperf* with respect to the weight as well as bias variables *X*. Individual variable is adjusted as per the gradient descent:

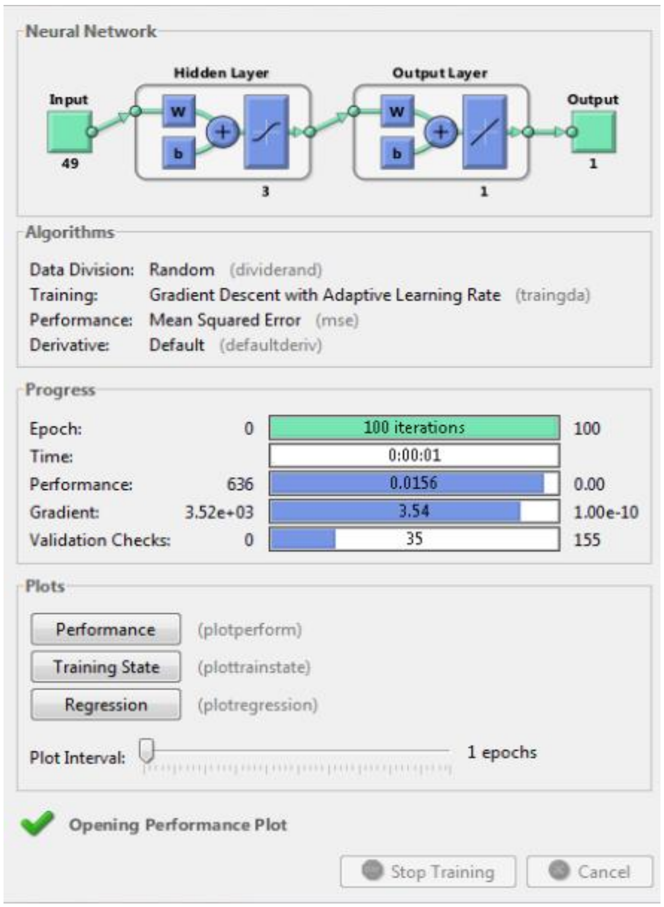
$$dX = lr*dperf/dX$$

At each epoch, the performance decreases towards the goal, then the learning rate is hiked by a factor of *lr_inc*. The performance is increased even higher than *max_perf_inc*, then learning rate is affected by *lr_dec*.

This is implemented as:

```
net = newff(p,t,3,{},'traingda');

net.trainParam.epochs = 100;     %Maximum number of epochs to train
net.trainParam.goal = 0;        %Performance goal
net.trainParam.showCommandLine  = 0;     %Generate command-line output
net.trainParam.showWindow   = 1;          %Show training GUI
net.trainParam.lr   = 0.05;             %Learning rate
net.trainParam.max_fail = 155;       %Maximum validation failures
net.trainParam.min_grad = 1e-10;      %Minimum performance gradient
net.trainParam.show  =  25;         %Epochs  between  displays  (NaN  for  no
displays)
net.trainParam.time = inf;            %Maximum time to train in seconds
```



Data Division: Random (dividerand)
Training: Gradient Descent with Adaptive Learning Rate (traingda)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

```
Epoch:                 0   100 iterations   100
Time:                      0:00:01
Performance:         636   0.0156           0.00
Gradient:       3.52e+03   3.54             1.00e-10
Validation Checks:     0   35               155
```

(plotperform)
(plottrainstate)
(plotregression)
Plot Interval: 1 epochs

Opening Performance Plot

Fig. 4.4a: Neural Network Training using traingda

The NN Training Tool clearly shows the neural network that is being trained and also the algorithms used for the same. Training state is also displayed.

The performance can be seen for each of the training, validation and test data sets. After the training of the network, best results were obtained on the validation data set. Other measures of how well the neural network has fit the data attributes are regression plot, error histogram.

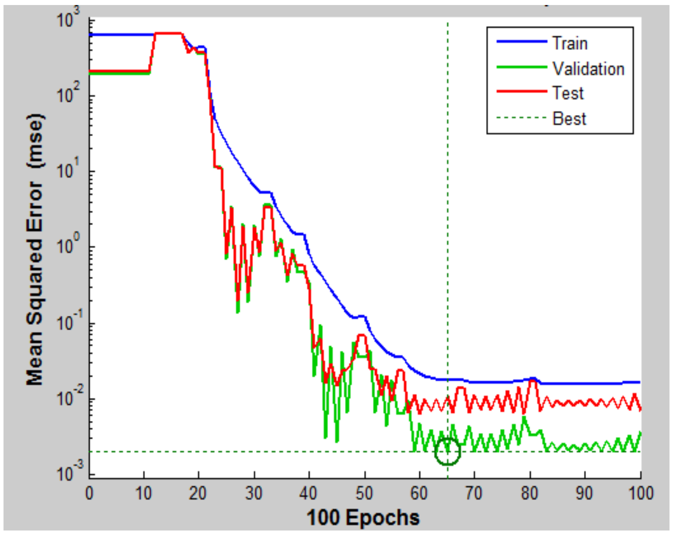


Fig. 4.4b: NN Training Performance (traingda)

It is observed from the fig. 4.4b that the best validation performance is achieved at 0.0019927 at epoch 65. Even if in such case, the data does not well fit the neural network, more training is suggested.

iv) traingdm: Gradient descent with momentum back propagation

traingdm can train any network as long as its weight, net input, and transfer functions have derivative functions. Back propagation determines derivatives of performance *perf* with respect to the weight and bias variables *X*. Each variable is adjusted as per the gradient descent with momentum,

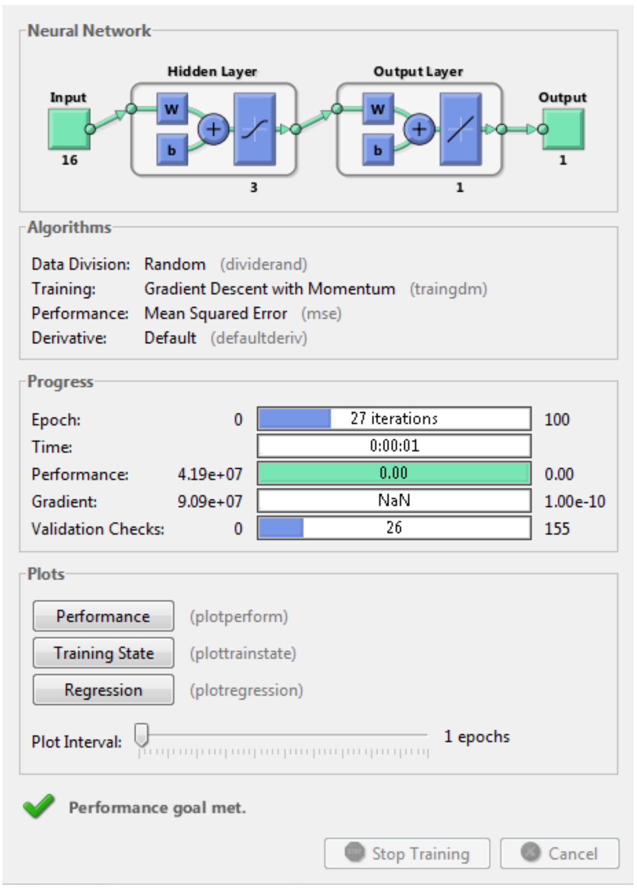
$$dX = mc*dXprev + lr*(1-mc)*dperf/dX$$

where *dXprev* is the previous alteration to the weight or bias.

This is implemented as:

```
net = newff(p,t,3,{},'traingdm');

net.trainParam.epochs = 100;     %Maximum number of epochs to train
net.trainParam.goal = 0;        %Performance goal
net.trainParam.showCommandLine  = 0;     %Generate command-line output
net.trainParam.showWindow   = 1;         %Show training GUI
net.trainParam.lr   = 0.05;            %Learning rate
net.trainParam.max_fail = 155;      %Maximum validation failures
net.trainParam.min_grad = 1e-10;     %Minimum performance gradient
net.trainParam.show  =  25;        %Epochs  between  displays  (NaN  for  no
displays)
net.trainParam.time = inf;          %Maximum time to train in seconds
```



Algorithms

Data Division: Random (dividerand)
Training: Gradient Descent with Momentum (traingdm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	27 iterations	100
Time:		0:00:01	
Performance:	4.19e+07	0.00	0.00
Gradient:	9.09e+07	NaN	1.00e-10
Validation Checks:	0	26	155

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

Performance goal met.

Fig. 4.5a: Neural Network Training using traingdm

Training using *traingdm* adds to the advantage that besides local gradient, it also acknowledges the variations in the surface of error. Using such function, the performance goal is met (Fig4.5a), as the best validation performance showed up at epoch 0 as 35284713.2542 (Fig. 4.5b).

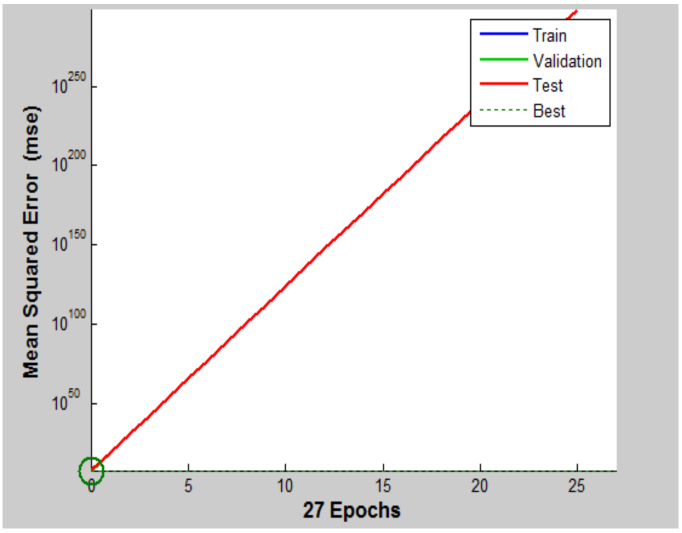


Fig. 4.5b: NN Training Performance (traingdm)

v) trainrp: Resilient back propagation

trainrp can train any network when its weight, network inputs, and transfer functions have derivative functions. Back propagation estimates derivatives of performance *perf* according to weight as well as the bias variables *X*. Individual variable is adjusted as:

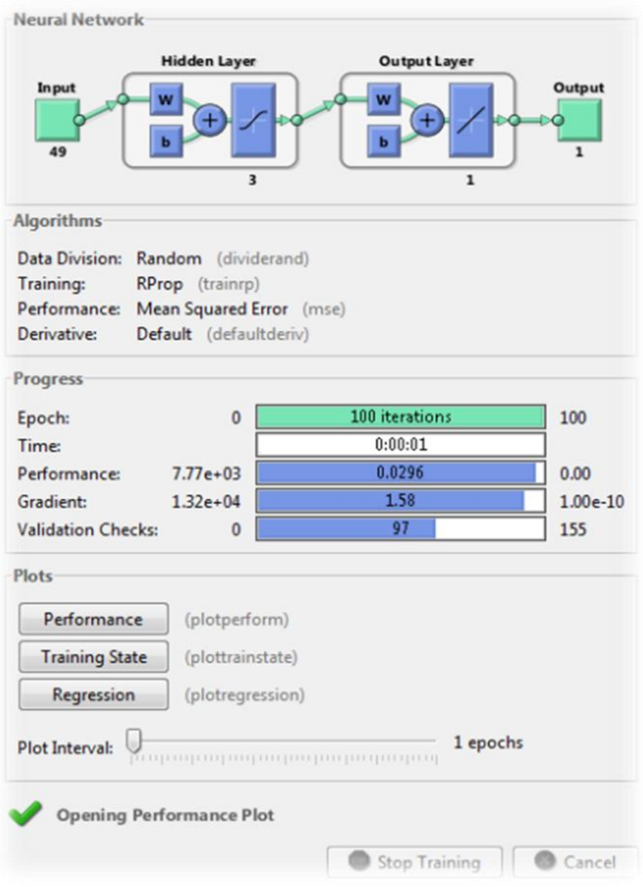
*dX = deltaX.*sign(gX);*

where the elements of *deltaX* are all set to *delta0*, with *gX* as the gradient.

This is implemented as:

```
net = newff(p,t,3,{},'trainrp');

net.trainParam.epochs = 100;     %Maximum number of epochs to train
net.trainParam.goal = 0;        %Performance goal
net.trainParam.showCommandLine  = 0;     %Generate command-line output
net.trainParam.showWindow   = 1;         %Show training GUI
net.trainParam.lr   = 0.05;            %Learning rate
net.trainParam.max_fail = 155;      %Maximum validation failures
net.trainParam.min_grad = 1e-10;     %Minimum performance gradient
net.trainParam.show  =  25;        %Epochs  between  displays  (NaN  for  no
displays)
 net.trainParam.time = inf;            %Maximum time to train in seconds
```



Algorithms

Data Division: Random (dividerand)
Training: RProp (trainrp)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	100 iterations	100
Time:		0:00:01	
Performance:	7.77e+03	0.0296	0.00
Gradient:	1.32e+04	1.58	1.00e-10
Validation Checks:	0	97	155

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Stop Training
Cancel

Opening Performance Plot

Fig. 4.6a: Neural Network Training using trainrp

The buttons at the bottom line of the NN training tool for various plots can be used after and even during training.

The resilient back propagation (Rprop) eliminates the side effects of the minute alterations in the weights and the biases due to small magnitude of the gradient. It has two additional parameters for initial weight change and maximum weight change from the other training functions discussed. This algorithm is quite quicker than the basic descent algorithm.

The performance of the Rprop is not much dependent on the arrangement of the training parameters. The best performance validation for *trainrp* is 0.0019927 at epoch 65 as illustrated in fig. 4.6b.

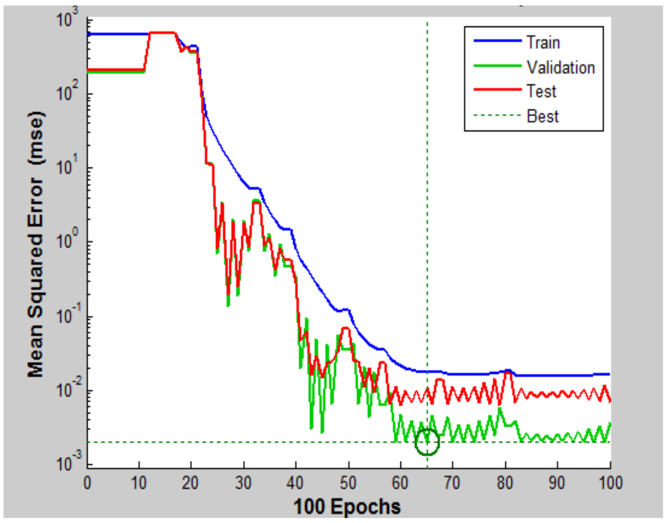


Fig. 4.6b: NN Training Performance (trainrp)

Well, the successful implementation of the proposed research methodology has been carried out with the help of MATLAB R2013a. The performance analysis of the various learning and back propagation algorithms used with the training of the feed forward artificial neural networks has been noted in terms of Mean Magnitude of Relative Error (MMRE).

Fig. 4.7 records the MMRE values obtained after the complete analysis of the training, testing as well as the validation state of the dataset fed into the neural network as performance criterion using the learning algorithmic functions.

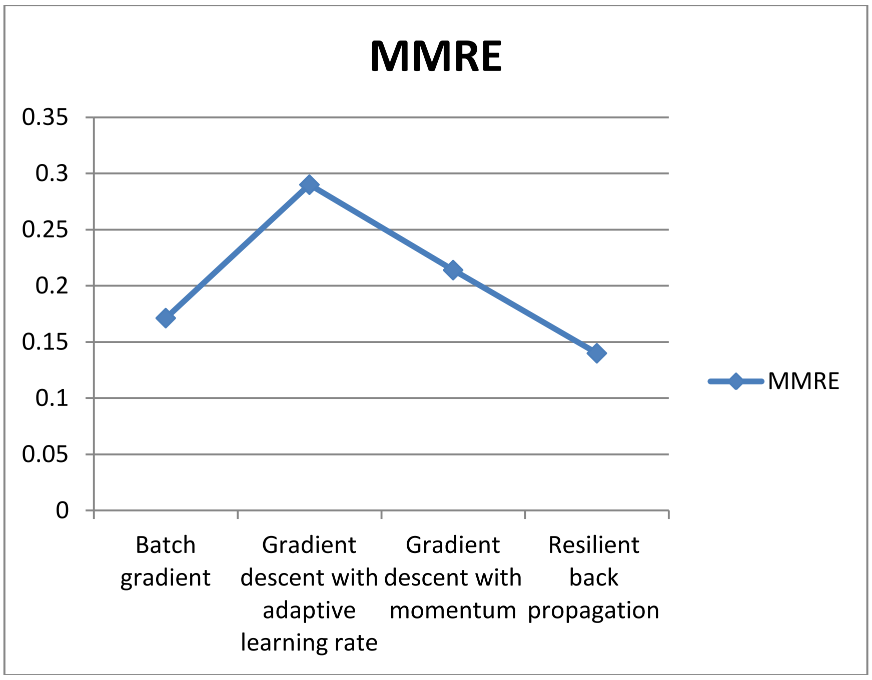


Fig. 4.7: MMRE Values of the algorithms used with artificial neural networks

Chapter 5
CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In the various training functions, feed forward neural networks are evaluated to predict the software effort for projects. The performances of the developed models were tested on the nasa software project dataset. It is concluded that neural networks are one of the most efficient approach for the function fit problems. The artificial neural network with the specified number of neurons can accurately fit any data set. Particularly, such networks are good at handling non linear data. So, the training functions model is able to provide good estimation for the software effort and cost using the neural network.

5.2 FUTURE WORK

The trained neural network provided efficient estimation capability for the software effort and the cost. This work can be extended as:

1. Explore the neural network approach for generating a suitable model structure for software metrics along software effort and cost estimation.
2. Different activation functions can be applied to the neural networks for quicker and more robust estimations.
3. More variant learning functions can be used to estimate the error rate and the efforts.
4. Advanced machine learning algorithms can be applied to the neural networks.

Chapter 6
REFERENCES

A. A. W. B. Barry Boehm (2007*), "Software Cost Estimation with COCOMO II"*

A. Idri, A. Abran and S. Mbarki (2006), *"An Experiment on the Design of Radial Basis Function Neural Networks for Software Cost Estimation"*

A.P. Engelbrecht (2006), *"Fundamentals of Computational Swarm Intelligence"* JohnWiley & Sons, New Jersey

Abeer Hamdy (2012), *"Fuzzy Logic for Enhancing the Sensitivity of COCOMO Model"*

Abran, A.; Gallego, J.J. (2009), *"Software Estimation Models and Economies of Scale"*

Amanjot Singh Klair & Raminder Preet Kaur (2012) *"Software Effort Estimation using SVM and kNN"*

Anjana Bawa, Mrs. Rama Chawala (2012), *"Experimental Analysis of Effort Estimation using Artificial Neural Network"*

Anupama Kaushik, AK Soni, Rachna Soni (2012) *"An Adaptive Learning Approach to Software Cost Estimation"*

B.W. Boehm et al *"The COCOMO 2.0 Software Cost Estimation Model"*, American Programmer, July 1996, pp.2-17.

Ch. Satyananda Reddy, KVSN Raju (2009) *"An Improved Fuzzy Approach for COCOMO's Effort Estimation using Gaussian Membership Function"*

Chadha, R., & Nagpal, S. Optimization of COCOMOII Model Coefficients using Tabu Search.

Ekbal Rashid, Vandana Bhattacherjee & Srikanta Patnaik (2012), *"The Application of Case-Based Reasoning to Estimation of Software Development Effort"*

Geeta Nagpal, Moin Uddin and Arvinder Kaur (2014), *"Grey Relational Effort Analysis Technique Using Regression Methods for Software Estimation"*

Gencel, C.; Buglione, L.; Demirors, O.; Efe, P. (2006); *"A case study on the evaluation of COSMIC-FPP and Use Case Points"*

Gharehchopogh, F. S., & Dizaji, Z. A. (2014). *A New Approach in Software Cost Estimation with Hybrid of Bee Colony and Chaos Optimizations Algorithms* (Vol. 2, No. 6, pp. 1263-1271). MAGNT RESEARCH REPORT.

Gharehchopogh, F. S., Maleki, I., & Khaze, S. R. (2014). A Novel Particle Swarm Optimization Approach for Software Effort Estimation. *International Journal of Academic Research, Part A, 6*(2), 69-76.

Gharehchopogh, F. S., Maleki, I., & Khaze, S. R. (2014). A Novel Particle Swarm Optimization Approach for Software Effort Estimation. *International Journal of Academic Research, Part A, 6*(2), 69-76.

Hamer P.G.,. Frewin,, G. D., "M.H. Halstead's Software Science – a critical examination", Proceedings of the 6th International Conference on Software Engineering, Sept. 13-16, 1982, pp. 197-206.

I. Myrtveit & E.Stensurd (1999), *"A controlled experiment to assess the benefits of estimation with analogy and regression models"*

Jagannath Singh & Bibhudatta Sahoo (2011), *"Software Effort Estimation with Different Artificial Neural Network"*

Jayashree K.M., Sathya Bama & J. Frank Vijay (2014), *"Software Effort Estimation Using Analogy"*

K. Ramesh & P. Karunanidhi (2013*), "Literature Survey On Algorithmic And Non Algorithmic Models For Software Development Effort Estimation"*

K.P. Manju & B.Arthi (2014), *"A Linear Regression Model with Exponential Transformation for Software Effort Estimation"*

Kamal, S., & Nasir, J. A. (2013). A Fuzzy Logic Based Software Cost Estimation Model. *International Journal of Software Engineering & Its Applications*, *7*(2).

Kavita Chaudhary (2010), *"GA Based Optimization of Software Development Effort Estimation"*

Kemerer, Chris F., and Michael W. Patrick (1993) *"Staffing factors in software cost estimation models"*

Khatibi, V., & Jawawi, D. N. (2011). Software Cost Estimation Methods: A Review 1.

Kowalska, J., & Ochodek, M. (2014). Supporting Analogy-based Effort Estimation with the Use of Ontologies. *e-Informatica Software Engineering Journal*, *8*(1).

M Jørgensen (2004), *"Top-down and bottom-up expert estimation of software development effort"*

Maitreyee Dutta and Prabhakar (2013), *"Prediction of Software Effort using Artificial Neural Network and Support Vector Machine"*

Maleki, I., Ghaffari, A., & Masdari, M. (2014). A new approach for software cost estimation with hybrid genetic algorithm and ant colony optimization.*International Journal of Innovation and Applied Studies*, *5*(1), 72-81.

Maryam Molani, Ali Ghaffari and Ahmad Jafarian (2014), *"A New Approach to Software Project Cost Estimation using a Hybrid Model of Radial Basis Function Neural Network and Genetic Algorithm"*

Mohd. Sadiq, Aleem Ali, Syed UvaidUllah, Shadab Khan and Qamar Khan (2013), *"Prediction of Software Project Effort Using Linear Regression Model"*

Muhammad Waseem Khan and Imran Qureshi (2014), "*Neural Network Based Software Effort Estimation: A Survey"*

P. Subitsha, J.Kowski Rajan (2014), *"Artificial Neural Network Models for Software Effort Estimation*

Prasad Reddy PVGD, Sudha KR, Rama Sree P and Ramesh SNSVSC (2010) *"Software Effort Estimation using Radial Basis and Generalized Regression Neural Networks"*

R. Banker, R. Kauffman, C. Wright, D. Zweig. (1994), "*Automatic Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineering (CASE) Environment", IEEE Transactions on Software Engineering*

R.M. Dawes, D. Faust and P.E. Meehl (1989), "*Clinical versus actuarial judgement"*

Rao, P. S., & Kumar, R. K. (2015, January). Software Effort Estimation through a Generalized Regression Neural Network. In *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India (CSI) Volume 1* (pp. 19-30). Springer International Publishing.

Ridhika Sharma (2013), *"Survey: Non Algorithmic Models for Estimating Software Effort"*

Rizvi, S., Abbas, S. Q., & Beg, R. AHybrid FUZZY-ANN APPROACH FOR SOFTWARE EFFORT ESTIMATION.

Seref, B., & Barisci, N. Software Effort Estimation Using Multilayer Perceptron and Adaptive Neuro Fuzzy Inference System.

Sriman Srichandan (2012), *"A new approach to Software Effort Estimation using Radial Basis Function Neural Networks"*

Sweta Kumari , Shashank Pushkar CSE & BIT Mesra, Ranchi India (2013), *"Performance Analysis of the Software Cost Estimation Methods: A Review"*

Symons, C. (1991), *“Software sizing and estimating: Mk II FPA (Function Point Analysis)”*, John Wiley & Sons, New York

Thayer, H.R. (2001), *“Software Engineering Project Management, Second edition IEEE CS Press”*

Tulin Ercelebi Ayyilidiz & Altan Kocyigit (2014), *“An Early Software Effort Estimation Method Based on Use Cases and Conceptual Classes”*

Usha Gupta and Manoj Kumar (2013), *“Software effort estimation through clustering techniques of RBFN network”*

Venus Marza and Mohammad Teshnehlab (2012) *“Estimating Development Time and Effort of Software Projects by using a Neuro-Fuzzy Approach”*

Ziauddin, Z., Shahid Kamal, S. K., Shafiullah khan, S. K., & Jamal Abdul Nasir, J. A. N. (2013). A Fuzzy Logic Based Software Cost Estimation Model.*International Journal of Software Engineering and Its Applications*, *7*(2), 7-18.

Zibigniew Michalewicz, Robert Hinterding & Maciej Michalewicz, *“Evolutionary Algorithms”*