Journal of Computer Sciences 1 (4): 505-509, 2005
ISSN 1549-3636

Bayesian Regularization in a Neural Network Model to Estimate Lines of Code Using Function Points

[1]K.K. Aggarwal, [1]Yogesh Singh, [1]Pravin Chandra and [2]Manimala Puri
[1]GGS Indraprastha University, Delhi, India, [2]IT Department, D.Y.Patil, COE, Pune, India

Abstract: It is a well known fact that at the beginning of any project, the software industry needs to know, how much will it cost to develop and what would be the time required ? . This paper examines the potential of using a neural network model for estimating the lines of code, once the functional requirements are known. Using the International Software Benchmarking Standards Group (ISBSG) Repository Data (release 9) for the experiment, this paper examines the performance of back propagation feed forward neural network to estimate the Source Lines of Code. Multiple training algorithms are used in the experiments. Results demonstrate that the neural network models trained using Bayesian Regularization provide the best results and are suitable for this purpose.

Key words: Neural network, estimation, lines of code, function point

INTRODUCTION

The estimation of resource expenditure (example, effort, schedule) is an essential software project management activity. Most projects (60-80 %). encounter effort and schedule overrun[1-5].

Software development involves a number of interrelated factors which can affect development effort and time and it is a complex dynamic process.

It is a challenge to estimate the lines of code for the project during the early stages of project as very little is known about the problem. Several researchers have suggested various techniques to predict software effort namely model based (SLIM, COCOMO Checkpoint), Expert based (Delphi) , Regression based etc. The latest of these techniques are machine learning techniques. There are a number of approaches[1,2,3] to machine learning namely Neural Networks, Fuzzy Logic, Case Based Reasoning and Hybrid Systems. Many researchers[6-9] have explored the possibility of using Neural Networks for estimating the effort. Neuro Fuzzy models[10] have also been explored and they are found to be useful in software estimation. This paper focuses on using a neural network to predict the lines of code when the function point, the FP standard used, the language one is going to use and the maximum team size is known. The ISBSG repository that is available for a number of projects is used to prove that neural networks are indeed suitable for this purpose. It also examines which training algorithm is best suited for the purpose.

Artificial neural networks: Artificial neural networks can model complex non-linear relationships and approximate any measurable function. They can be used as an effective tool for pattern classification and clustering[11,12]. They are particularly useful in problems where there is a complex relationship between an input and output. It has been established that a one hidden layer feedforward network with (sufficient number of) sigmoidal nodes can approximate any continuous mapping with arbitrary precision[13-16]. The feed forward multi layer network is a network in which no loops occur in the network path. A learning rule is defined as a procedure for modifying the weight and biases of a network with the objective of minimizing the mismatch between the desired output and the obtained output from the network for any given input. The learning rule / network training algorithm is used to adjust the weights and biases of the network in order to move the network outputs close to the targets. The classical backpropagation algorithm was the first training algorithm developed[17]. The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly - the negative of the gradient[17] though second order optimization algorithms like the conjugate gradient, the Levenberg-Marquardt and Baysian learning algorithms have also been developed. In this paper, a four input and one output network is used. The network uses only one hidden layer. The activation functions at the hidden layer and the output layers are the tangent - hyperbolic (*tanh*) function. The network inputs are (a) The function point count for projects, (b) the team size, (c) the level of the language used in development and (d) the function point standard. The block diagram of the network used is shown in Fig. 1.

Figure 1 shows a model whose inputs are function points, language used, FP standard and maximum team size. The output (target) is lines of code.

Neural Network model for estimating lines of code:

Corresponding Author: Manimala Puri, IT Department, D.Y.Patil, COE, Pune, India

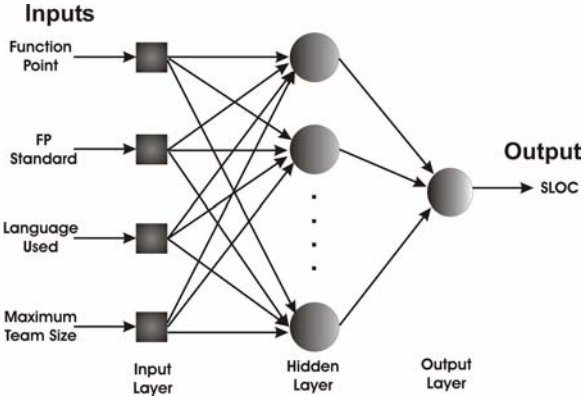

Fig. 1: Neural network model

EXPERIMENT

Study area and Data used: The project data used was that of International Software Benchmarking Standards Group (ISBSG) repository data (release 9). Out of the various fields available, the following fields were used as inputs.

Project ID: This was used for identifying projects

Functions points: The function points count for that particular project.

F.P. standard: This field specifies which function point standard was used e.x. CPM 4.0 IFPUG 4, IFPUF4.1 etc.

Language: This defines the language type used for the project e.g. 3GL, 4GL, Application Generator etc.

Lines of code: The number of the source line of code (SLOC) produced by the project. This is not available for all projects.

Since SLOC is not available for all projects , only those projects were considered for the experiment where SLOC data was available . This lead to a data set of 88 projects.

MATERIALS AND METHODS

The function point count, function point standard, language used a nd maximum team size were used as inputs. Outliers were removed from all data sets. The function point data was normalized by linear scaling between -1 and 1. The FP standard was coded as shown in table I and the language used, was coded as shown in Table 2. Since there was only one data POINT pertaining to 5 GL that was dropped. All data variables are scaled in the range -1 to 1.

Table 1: Input codes for FP Std

FP Std.	Code
CPM 4.0	0.5
IFPUG 4	0.25
IFPUG 4.1	-0.25
Backfired	-0.5

Table 2: Input codes for language used

Language used	Code
3 GL	0.25
4 GL	0.5

The neural network used a sigmoid feed forward network with a single hidden layer using the neural network tool for of MATLAB. Seventy one exemplars were used for training with SLOC as the target. The neurons in the hidden layer were varied from five to sixteen. It was found that the ensemble with fifteen neurons in the hidden layer yielded best results.

Thus there are four nodes in the input layer, fifteen neurons in the hidden layer and one node in the output layer. The MATLAB adaptation learning function selected for this experiment was 'learngdm, the performance function used was mean square error (MSE). Transfer functions used were, tangent - hyperbolic in both the hidden and the output layers. The goal was kept as 0.00(though it was never achieved, but a goal of 10^{-16} was reached which is as good as 0.)The no. of epochs was kept as 1000. After training, testing was done on the network from the data set of seventeen projects. Random partioning was done to form three sets of training and test data. After training, testing was done and the output obtained were compared with the target values. In this case we obtained seventy one training cases and seventeen test cases in every set. The objectives of the experiments were twofold:

* To verify if neural networks can be used for prediction of SLOC counts on the basis of Function Points, team size, function point standard and language type used in development.
* To empirically evaluate the training algorithms and to find which training algorithm is suitable for the estimation purpose.
* The experiment was conducted using the same neural network but using different algorithms as shown in Table

After performing the same experiment with different algorithms, the results are compared in the next section.

Error measurements: Different error measurements have been used by various researchers. We have chosen the mean absolute Percentage Error(MAPE) MAPE is calculated as follows[7]

Table 3: Different training algorithms

trg. Fnc.	Description
trainb	trains a network with weight and bias learning rules with batch updates. The weights and biases are updated at the end of an entire pass through the input data.
Trainbfg	updates weight and bias values according to the BFGS quasi-Newton method.
Trainbr	updates the weight and bias values according to Levenberg-Marquardt optimization. It minimizes a combination of squared errors and weights and then determines the correct combination so as to produce a network that generalizes well. The process is called Bayesian regularization.
Trainc	trains a network with weight and bias learning rules with incremental updates after each presentation of an input. Inputs are presented in cyclic order.
train cgb	updates weight and bias values according to the conjugate gradient backpropagation with Powell-Beale restarts.
train cgf	updates weight and bias values according to the conjugate gradient backpropagation with Fletcher-Reeves updates.
train cgp	updates weight and bias values according to the conjugate gradient backpropagation with Polak-Ribiere updates.
train gd	updates weight and bias values according to gradient descent.
train gda	updates weight and bias values according to gradient descent with adaptive learning rate.
train gdm	updates weight and bias values according to gradient descent with momentum.
train gdx	updates weight and bias values according to gradient descent momentum and an adaptive learning rate.
train lm	updates weight and bias values according to Levenberg-Marquardt optimization.
train oss	updates weight and bias values according to the one step secant method.
train rp	updates weight and bias values according to the resilient backpropagation algorithm (RPROP).
train scg	updates weight and bias values according to the scaled conjugate gradient method.

Table 4: Results using various training algorithms

At	Run 1				Run 2				Run 3			
Algorithm	MAPE	Co-relation	Std	Significance	MAPE	Co-relation	Std	Significance	MAPE	Co-relation	Std	Significance
Train gd	17.35	0.33	0.28	0.81	13.18	0.48	0.17	0.95	18.57	0.03	0.23	0.10
Train gdm	20.50	0.20	0.18	0.57	14.76	0.42	0.16	0.92	22.04	-0.32	0.21	0.81
Train gda	17.97	0.33	0.35	0.80	15.76	0.41	0.14	0.94	19.89	0.09	0.32	0.28
Train gdx	24.32	0.56	0.31	0.98	16.44	0.41	0.15	0.91	22.95	0.37	0.31	0.87
Train rp	18.42	0.46	.21	0.94	16.83	0.34	0.15	0.84	15.44	0.67	0.19	0.99
Train oss	34.59	0.35	0.56	0.83	17.02	0.50	0.16	0.96	34.05	0.61	0.38	0.99
Train scg	39.20	0.27	0.63	0.71	25.63	0.31	0.31	0.79	56.14	0.63	0.63	0.99
Train cgp	19.42	0.39	0.25	0.88	17.12	0.54	0.16	0.98	63.72	0.52	0.72	0.97
Train cgf	24.91	0.69	0.32	0.99	23.09	0.44	0.27	0.93	39.17	0.60	0.48	0.99
Train cgb	25.75	0.39	0.36	0.87	23.97	0.75	0.24	0.99	72.80	0.36	0.80	0.86
Train b	20.63	0.20	0.18	0.56	12.85	0.49	0.15	0.96	23.37	-0.09	0.31	0.28
Train bfg	30.84	0.08	0.53	0.26	33.79	0.64	0.41	0.99	50.61	0.73	0.63	0.99
Train lm	37.50	-0.22	0.46	0.62	89.16	0.24	0.93	0.67	63.59	0.55	0.75	0.98
Train br	13.93	0.71	0.14	0.99	12.94	0.60	0.13	0.99	17.08	0.70	0.19	0.99

$$MAPE = \left(\sum_{j=1}^{j=n} \left[\frac{Estimate - Actual}{Actual} \right] \right) \div n \times 100 \qquad (1)$$

If MAPE is small, the better is the model and the predictions are a good set of predictions.

The Correlation Coefficient (r). or correlation coefficient for short is a measure of the degree of linear relationship between two variables. The correlation coefficient may take on any value between plus and minus one

Significance of Correlations.(sig). The significance level calculated for each correlation is a primary source of information about the reliability of the correlation

Standard Deviation.(std.). The standard deviation (this term was first used by Pearson, 1894) is a commonly-used measure of variation. The standard deviation of a population of values is computed as:

$$\sigma = [\Sigma (x_i - \mu)^2 / N]^{1/2} \qquad (2)$$

where:

μ is the population mean

N is the population size.

RESULTS AND DISCUSSION

The results using various training algorithms are as shown in Table 4.

Results demonstrate that *train br* algorithm can be rated as the best. average MAPE in this case is 14.65, average co-relation is 0.64 and average significance is 0.99 . *train gd* algorithm can be rated as the next best one with average mape as 16.36, average co-relation as 0.28 and average significance as 0.62.these are followed by *train rp* algorithm with average values of MAPE, co-relation and significance being 16.89, 0.49 and 0.90 respectively. The plots of actual SLOC and predicted SLOC for various test projects using the train br algorithm for three runs are as shown in Fig. 2- 4.

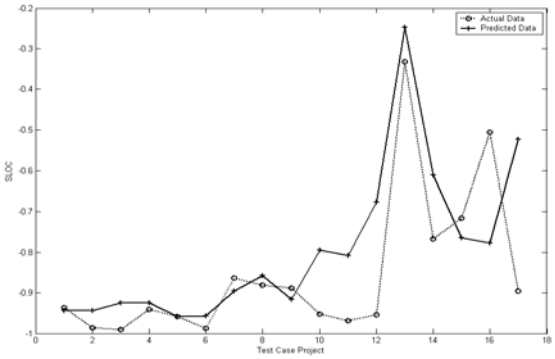

Fig. 2: SLOC (actual and predicted) Vs. Projects for RUN 1

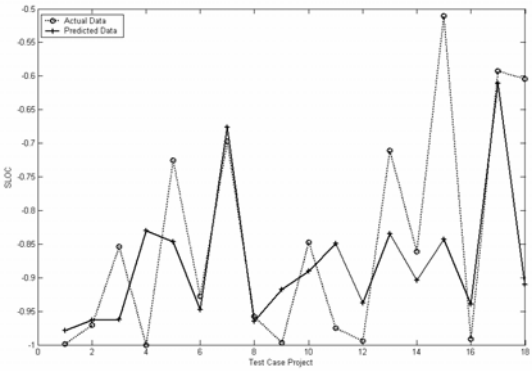

Fig. 3: SLOC (actual and predicted) Vs. Projects for RUN 2

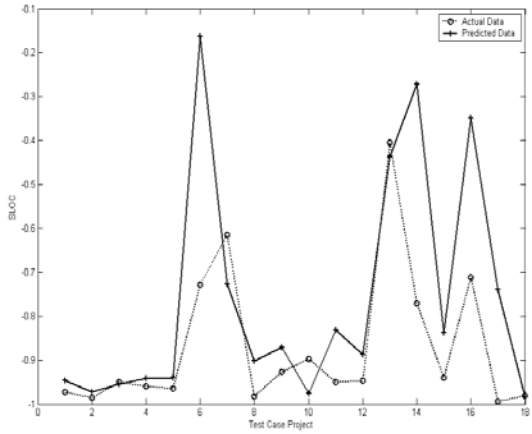

Fig. 4: SLOC (actual and predicted) Vs. Projects for RUN 3

Future scope: The neural network model used here could further be extended to a neural fuzzy model being trained and tested for the same ISBSG data (Release 9). There is a possibility that this model could be a better model.

CONCLUSION

In the present work, the possibility of use of neural networks for estimating lines of code for software project was explored. The ISBSG Data(Realease 9) , was used to train and test the neural network.

It is concluded from experimental work the neural networks can be very well used for estimating the lines of code once the function point count is known. Also it is concluded that the train br algorithm yields the best results.

REFERENCES

1. Aggarwal, K.K. and Yogesh Singh, 2001. Software Engineering Programs, Documentation Operating Procedure, New Age International Publishers.

2. Pressman, 1997. Software Engineering. A Practitioners Approach. McGraw Hill.
3. Sommerville, R., 1996. Software Engineering. Addison Wesley.
4. Kjetil Molekkan and Magne Jergense, 2003. A review of surveys on software effort estimation. Proc. 2003 Intl. Symp. Empirical Software Engineering.
5. Putnam, L.H., 1978. A general empirical solution to the macro software sizing and estimation problem. IEEE Trans. Software Engineering, 4: 345-361.
6. Dawson, C.W., 1996. A neural network approach to software projects effort estimation. Trans. Information and Commun. Technol.
7. Finnie, G. and G. Wittig, 1996. AI tools for software development effort estimation. IEEE Trans. Software Engineering, pp: 346-353.
8. Ali Idri Tagi, M. Khoshgoftaar and Alin Abran, 2002. Can neural networks be easily interpreted in software cost estimation. IEEE Trans. Software Engineering, pp: 1162-1167.
9. Fuzzy systems and neural networks in software engineering project management. J. Applied Intell., 4: 31-42.
10. Hodgkinson, A. and P. Garatt, 1999. A neuro fuzzy cost estimator. Proc. Intl. Conf. Software Eng. Application, pp: 401-406.
11. Haykyn, S., 2003. Neural Networks, A Comprehensive Foundation. Prentice Hall, India.
12. Agarwal, K., Y. Singh and M. Puri, 2005. Measurement of software understandability using neural networks. Proc. Intl. Conf. Multidimensional Aspects of Engineering, IEEE, WEI Group.
13. Cybenko, G., 1989. Approximation by superposition of a sigmoidal function. Math, Control, Signal and Syst., 5: 233-243.
14. Funahashi, K., 1989. On the approximate realization of continuous mappings by neural networks. Neural Networks, 2: 183-192.
15. Barron, A.R., 1993. Universal approximation bounds for superposition of a sigmoid function. IEEE Trans. Inform. Theory, 39: 930-945.
16. Hornik, 1989. Stinchcombe and white, multilayer feedforward networks are universal approximators. Neural Networks, 2: 359-366.
17. Rumelhart *et al*., 1986. Learning representations by back –propagating errors. Nature, 323: 533-6.