Transactions on Information and Communications Technologies vol 20, © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

An Artificial Neural Network Approach to Software Testing Effort Estimation

Christian W Dawson
School of Mathematics and Computing, University of Derby, Kedleston Road, Derby, DE22 1GB, UK
EMail: C.W.Dawson@Derby.ac.uk

Abstract

Trying to predict the effort needed to test prewritten software is a complex problem as the amount of work involved in software testing depends on a number of independent *and* related factors (for example, lines of code, unit complexity etc.). Artificial neural networks appear well suited to problems of this nature as they can be trained to understand the explicit and inexplicit factors that drive a test's cost. For this reason, artificial neural networks were investigated as a potential tool to improve software testing effort estimation using project data supplied by Rolls-Royce and Associates Limited. In addition, in order to deal with uncertainties that exist in modelled results, statistical analyses were employed to identify confidence intervals for predicted costs. This paper discusses these analyses and comments on the results that were obtained when artificial neural networks were developed, trained and tested on the data supplied.

1 Introduction

Although there has been a significant amount of research in software measurement, software metrics and software project estimation (for example, see texts such as [1, 2]), there has been a limited amount of work focused towards the prediction of software testing effort. This form of effort estimation is particularly important for those companies who provide software testing services. Rolls-Royce and Associates Limited are one such company who test software developed by other organisations. In this case the software tested is used in aircraft jet engines. Their customer initially supplies information on the software units they require testing (for example, the unit's size in terms of lines of code) and a response is made as a tender for the work. Currently, this tender is based on a best estimate of the effort required. This estimate is somewhat subjective and results, presented later in this note, show how these estimates can be significantly improved by employing an artificial intelligence technique such as artificial neural networks (ANNs). In addition, the results of the ANN approach are compared with a standard statistical model.

1.1 Unit Testing Effort Estimation

Unfortunately, cost estimation in software development is not an exact science, as there are a number of qualitative factors involved that can rarely be explicitly identified. Because of this, several techniques and approaches have been proposed for estimating software development effort, and the quantity of literature on the subject is enormous. Without an ability to measure, or accurately estimate, a software project's characteristics, organisations have little management control over that project's behaviour, risks are not clearly identified, and potentially disastrous projects may be undertaken. In fact '15% of software projects are not completed due to grossly inaccurate estimates of development effort' [3].

Usually, software project estimates are based on initial estimates of the product characteristics and the process that is being adopted. For example, product estimates might include quantitative measures such as product size (in terms of lines of code measured by an agreed standard), functionality (in terms of function points), and qualitative measures such as complexity and quality. Process estimates might include factors such as team productivity, effectiveness of defect removal procedures and so on.

Techniques that are often used for software project estimates include COCOMO (which uses an algorithmic approach based on product estimates [4]), Delphi (a technique used to balance different experts' opinions [5]), and function points (a technique that measures the complexity of a product [6]).

Using artificial neural networks to predict software development effort is not a new idea. For example, Kumar et alia [7] used neural networks to estimate manpower buildup levels in software projects, based on task concurrency, inverse application complexity, and schedule pressures. In addition, Hakkarainen et alia [8] used artificial neural networks to predict software product size. However, trying to predict the costs involved in unit testing is a relatively immature field. Applying ANNs to this area has yet to be fully explored.

2 Artificial Neural Networks

2.1 Overview

Artificial neural network research tends to fall into two main categories; first is the development and improvement of existing ANN training algorithms and topologies. The second area involves the application and evaluation of existing ANN research to complex problems. This is the category into which this particular study falls. Existing ANN algorithms have been adopted and adapted to the problem of software testing effort estimation. The networks chosen for this study were feed forward, three-layered networks, trained using a modified backpropogation algorithm.

The three layers in this case are an *input* layer, which distributes inputs to the *hidden* layer, and an *output* layer. The number of nodes used in the input layer and output layer are determined by the problem to which the network is being applied. The number of nodes used in the hidden layer is chosen by the user and there are no clear rules as to how many nodes one should employ in this layer. Consequently, different numbers of hidden nodes were evaluated in this study in an attempt to identify the 'best' ANN configuration for software testing effort prediction.

Explanation of ANN structure and discussion of the backpropogation training algorithm is beyond the intended scope of this paper as there are numerous articles and texts devoted to these issues. For more information on ANNs the reader is directed towards texts such as Wasserman [9] and Gallant [10].

In order to implement and evaluate ANNs in this study, four computer programs were developed. The first program is used to create the training sets from data of previous projects. The second program standardises the data within a training set before it is passed to a third program that generates and trains the ANNs. The backpropogation algorithm used for training the ANNs was adapted to include *momentum* [10] and a dynamically controlled step size algorithm [11]. The fourth program is used to test a trained ANN. This testing can include Monte Carlo simulation of the network to provide estimates based on initial best guesses by the user. The distribution functions generated by this program, as text files, can then be analysed to determine the uncertainties involved with the estimated costs [11].

3 Data Acquisition and Analyses

3.1 Overview

As with the development of any model, one is always limited by the quantity and quality of data that are available. Although, in this case, the quality of data available were high, the quantity of data supplied were somewhat limited, although there were sufficient data available for this preliminary study.

Data were obtained for a recently completed test project. This project consisted of 12 *objects* each containing from 2 to 19 individual *units* for testing. In total there were 89 units in the project with testing durations ranging from 1 to 98 hours. Total testing cost for the entire project was 1546 hours (cost and duration being synonymous). Table 1 provides a typical extract from the data supplied. In this table data are presented for Owner (who is responsible for testing the unit), Package (object name), Unit name, Actual Duration (actual duration of each unit test in hours), McCabe's Complexity Metric, Number of Lines of ADA Code (in each unit), and Estimated Duration (estimated duration of unit test in hours).

Owner	Package	Unit	Actual Duration	McCabe Complexity	Lines of ADA	Estimated Duration
AB	1	A	36	14	125	45
CD	2	B	11	7	50	30

Table 1 Extract of the Data Supplied

Using statistical analyses three factors (or *drivers*) were identified from the data supplied as having an influence on a unit's test cost; lines of code in the unit, McCabe's complexity metric, and number of units within an object. These three factors were used as the three inputs into an ANN with actual duration being used as the output.

3.2 Neural Network Training

There are a number of different parameters one can manipulate when constructing ANNs. These parameters include variations to the number of hidden nodes one chooses to have in a network, variations to the 'speed' of the training algorithm employed (by adjusting a *learning parameter*), and variations to the length of time one chooses to train a network for (the number of training cycles or *epochs*). In this particular study a number of different network configurations were investigated (different numbers of hidden nodes) along with various different training cycles. The results of the more accurate configurations are presented later.

In order to test the ANNs in this study it was necessary to split the data into two sets; that which could be used for training (the *calibration* set) and that which could be used for testing (the *validation* set). The calibration and validation data sets were chosen to be mutually exclusive and jointly exhaustive. It was decided that a sufficiently rigorous test for the ANNs would be train them on 90% of the data and test their ability to predict previously unseen unit test efforts on the remaining 10% of the data. This approach was repeated ten times so that the ANNs were forced to predict every unit's test duration without having being exposed to that unit beforehand.

4 Evaluation

4.1 Comparative Statistical Model

In order to evaluate the effectiveness of the ANN approach in this study, results were compared with existing best estimates and with a more conventional statistical model. In this case the conventional statistical model employed was a stepwise multiple linear regression (MLR) approach. This technique identified the most significant factors affecting unit testing cost and reduced the affects of multicollinearity between these drivers using a stepwise procedure. It was found that there was a close correlation between the lines of code of the software units under test and McCabe's complexity measure of those units. Consequently, the McCabe measure was excluded by the stepwise technique from the statistical model produced. The MLR model was tested in the same way as the ANN approach and the comparative results are presented below.

4.2 Measuring Accuracy

There are a number of measures one can use to determine the overall accuracy of modelled results. In this study two such measures were employed; the mean squared error (MSE) and the mean squared relative error (MSRE):

$$\mathrm{MSE} = \frac{\sum_{i=1}^{n} (T_i - \hat{T}_i)^2}{n} \tag{1}$$

$$\mathrm{MSRE} = \frac{\sum_{i=1}^{n} \frac{(T_i - \hat{T}_i)^2}{T_i}}{n} \tag{2}$$

where T_i are the n modelled durations and T_i are the n actual durations.

Squared errors provide a good indication of errors made for long durations, whilst relative errors provide 'a more balanced perspective of the goodness of fit' for shorter durations [12]. In addition, the coefficient of determination, r^2, was calculated for each model to measure the proportion of the variability of duration accounted for by the model. The absolute error was also calculated to determine how closely the modelled durations approached the overall project total. A negative total error implied an overestimate of a project's duration. Percentage total errors were also calculated as total error divided by total project duration.

4.3 Results

Table 2 provides a comparative overview of the results that were obtained on the representative project. The ANNs presented in this table were all trained for 50000 epochs, which gave the best overall predictions. As can be seen from this table all ANNs significantly out perform the existing approach to estimation. The existing approach overestimates existing durations by as much as 18% whereas ANN2 is accurate to within about 1% of the project's total duration.

	MSE (Hours^2)	MSRE	Total Error (Hours)	% Error	r^2
Current Approach	265	9.7	-284	-18%	0.191
ANN1	222	3.79	-68	-4%	0.378
ANN2	323	4.88	+11	1%	0.249
ANN3	419	5.60	-64	-4%	0.195
Step wise MLR	233	10.14	+31	2%	0.297

Table 2 Comparative Results of Different Techniques (ANN1 - 2 hidden nodes, ANN2 - 4 Hidden nodes, ANN3 - 8 Hidden nodes)

Table 2 also highlights how existing techniques could be improved by employing a simple statistical approach such as the multiple linear regression approach discussed earlier. In this case, using a stepwise MLR approach, percentage error results are comparable with those of the trained ANNs. Some measures, such as the MSE and coefficient of determination, show that better estimates could be made using this approach than the trained ANNs.

Figure 1 shows a comparison between the MLR approach and ANN1. This figure plots predicted duration against actual duration. A perfect model would produce a straight line with all points lying on $f(x) = x$ (shown by the bold solid line). As can be seen, both models appear to capture the general trend of the durations but their results are by no means perfect. The coefficient of determination (r^2) provides a useful measure of this relationship. In the case of the MLR model, r^2 is 0.297 and in the ANN1 model, r^2 is 0.378.

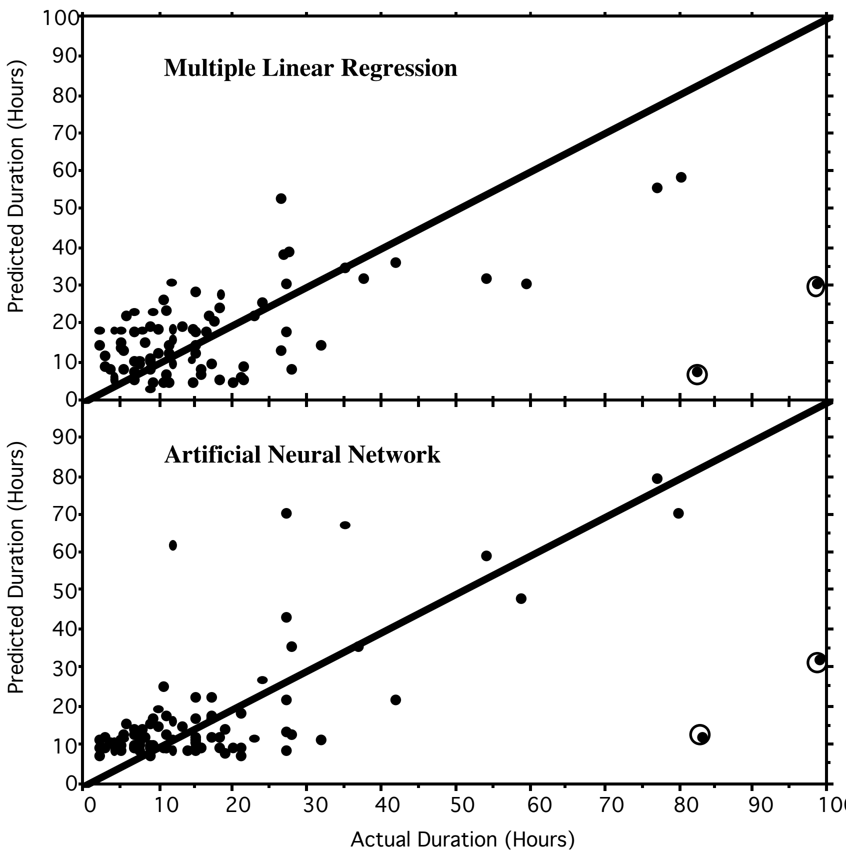

Figure 1 Comparison of ANN1 and MLR Models' Predictions

It is worth noting the two circled estimates in Figure 1 for both models. In both cases the MLR and ANN□models have grossly underestimated the actual duration of each unit's testing duration. An inspection of the data confirmed that these inaccurate predictions were by no means unexpected as they both occurred in rather insignificant software units. Clearly, some other factors (for example, Owner) were having an affect on the time taken to test these two units; factors which had not been included in the models.

The MLR model showed somewhat variable predictions at lower durations compared with the ANN model. This is highlighted in Figure 1 by the spread of points at lower durations and the higher MSRE measure shown in Table 2.

4.4 Confidence Limits

A useful outcome from the above evaluation was determining the confidence one has in a model's predictive powers. Using the standard error of the least squares regression line between predicted duration and actual duration (ie this time the regression of actual duration on predicted duration), one can provide confidence intervals for each model's predictions. For example, in the ANN1 model, the standard error about the regression line was calculated as 14.35 hours. Thus, for an ANN1 prediction of 20 hours, one is 68% confident (one standard error about the regression line) that the actual duration of the unit test the ANN1 model is attempting to predict, is actually between 18.5±14.35 hours (where 18.5 is calculated from the regression model). Clearly, the more accurate the model is, the shorter and more valuable this confidence interval becomes as, in this case, the confidence interval is so broad as be somewhat meaningless.

Unfortunately, due to hetroscedasticity, these confidence intervals are only valid within a certain range. The level of hetroscedasticity is also difficult to determine. In the case of ANN1, from a simple visual inspection of the data, the homoscedastic range appeared to be limited to durations up to approximately 30 hours. Predicted durations over 30 hours appeared too variable to enable confidence limits, based on the standard error about the regression line, to be accurately determined. For predicted durations of up to 30 hours, the standard error about the regression line is 6.6 hours. Thus, for a prediction of 20 hours, the 68% confidence interval is now 12.9±6.6 hours within the homoscedastic range.

6 Conclusions

This note has shown that an ANN model can be valuable for predicting software testing effort. The results clearly show improvements on existing 'best guesses' and are, in some cases, better than standard statistical models.

Analyses of the models also provides additional useful information. In this case, confidence intervals for each model's predictions were calculated. However, unless the accuracy of a model is reasonable, and predicted results are relatively homoscedastic, the results a model produces can be so vague as to be somewhat meaningless.

Clearly, further research is required to determine an optimum ANN configuration (topology, training rate, epochs and so on) for problems of this nature. As with all such models, improvements can continue to be made as more data become available and ANNs are able to learn from a larger pool of previous project histories.

7 References

[1] Fenton, N.E. *Software Metrics, A Rigorous Approach*, Chapman and Hall, London, 1991.

[2] Kan, S.H. *Metrics and Models in Software Quality Engineering*, Addison-Wesley Publishing Company, USA, 1995.

[3] Jones, C. *Programming Productivity*, McGraw Hill, New York, 1986.

[4] Boehm, B.W. *Software Engineering Economics*, Prentice Hall, USA, 1981.

[5] Helmer-Heidelberg, O. *Social Technology*, Basic Books, New York, 1966.

[6] Albrecht, A.J. Measuring Application Development Productivity, *Proceedings of the IBM SHARE/GUIDE Applications Development Symposium*, Monterey, CA, October, pp. 83-92, 1966.

[7] Kumar, S. Krishna, B.A. & Satsangi, P.S. Fuzzy Systems and Neural Networks in Software Engineering Project Management, *Journal of Applied Intelligence*, **4**, pp. 31-52, 1994.

[8] Hakkarainen, J. Laamanen, P. & Rask, R. 'Neural Networks in Specification Level Software Size Estimation', *Proceedings 26th International Conference on Systems Sciences*, Wailea, 5-8 January, IEEE, **4**, pp. 626-634, 1993.

[9] Wasserman, P.D. *Neural Computing Theory and Practice*, Van Nostrand Reinhold, New York, 1989.

[10] Gallant, S.I. *Neural Network Learning and Expert Systems*, Massachusetts Institute of Technology, USA, 1994.

[11] Dawson, C.W. 'A Neural Network Approach to Software Project Effort Estimation', *Applications of Artificial Intelligence in Engineering*, **1**, pp. 229 - 237, 1996.

[12] Karunanithi, N. Grenney, W.J. Whitley, D. and Bovee, K. 'Neural Networks for River Flow Prediction', *Journal of Computing in Civil Engineering*, **8**(12), pp. 201 - 220, 1994.