USING ARTIFICIAL NEURAL NETWORKS AND FUNCTION POINTS TO ESTIMATE 4GL SOFTWARE DEVELOPMENT EFFORT

G.E. Wittig and G.R Finnie
School of Information Technology
Bond University
Gold Coast, Queensland 4229

ABSTRACT

The value of neural network modelling techniques in performing complicated pattern recognition and non-linear estimation tasks has been demonstrated across an impressive spectrum of applications. Software development is a complex environment with many interrelated factors affecting development effort and productivity. Accurate forecasting has proved difficult since many of these interrelationships are not fully understood. An attempt to capture the significant attributes of the software development environment to enable improved accuracy in forecasting of development effort is made using backpropagation artificial neural networks. The data for this study was gathered from commercial 4GL software development projects, across a large range of sizes. As is typical of software developments, the range in productivity and other development factors in the data set is also large, accentuating the estimation problem. Despite these difficulties the neural network model predictions were reasonably accurate in comparison with other published results, indicating the potential of the use of this approach.

INTRODUCTION

Reliable prediction of size and effort in software development projects is a necessary prerequisite to developing reliable cost and schedule estimates (de Marco, 1982). The size and development effort measures, such as function points or lines of code developed per person-month, act as technical productivity and performance indicators that facilitate the tracking and control of software developments.

Unfortunately it is by no means easy to estimate development effort accurately and much effort has been expended to develop metrics which attempt to measure size and complexity of programs and systems (Conte et al, 1986). Many of the early metrics were developed in the procedural third generation language (3GL) environment. The mainly non-procedural fourth generation language (4GL) developments, with their high level query languages, screen writers, report generators, and application generators are substantially different (Martin, 1982).

Function Point Analysis (FPA), which is technology independent (Dreger, 1989; Rudolph, 1986), has been used for both 3GL and 4GL developments to estimate development effort. An analysis of some commercial 4GL software developments from several organisations indicates the presence of large variations in productivity which must be accommodated to accurately estimate development effort (Ferens & Gurner, 1992; Wittig, 1991).

Interrelationships between the various factors affecting development effort are complex, not fully understood and have made development cost estimation difficult and sometimes inaccurate (Kemerer, 1987). These factors affecting development effort should not be analysed individually in isolation (Kitchenham, 1992), as it is necessary to ascertain their combined effect. Intuitively for example the proportionate difference in total development hours between a system developed by an expert programmer team and an average programmer team is significantly influenced by the complexity of the system being developed. In an analysis of 65 cases (Banker et al, 1991) to identify factors influencing maintenance productivity, the technique of Stochastic Data Envelopment Analysis was used and the authors noted that linear models are not likely to make adequate representations of the development process.

The value of neural network modelling techniques in performing complicated pattern recognition and non-linear estimation tasks has been demonstrated across an impressive spectrum of applications (White, 1988). A recent survey (Rumelhart et al, 1993) shows that the recent growth of neural network applications has been quite remarkable. It was just four years ago that the only widely reported commercial application outside the financial industry was the airport baggage explosive detection system. Since that time scores of industrial and commercial applications have become known. A few of these applications include telecommunications, particle accelerator beam control, credit card fraud detection, machine and hand-printed character recognition, cursive handwriting recognition, mass spectra classification, quality control in manufacturing, petroleum exploration, war on drugs, medical applications, financial forecasting and portfolio management, and loan approval. The details of many are considered corporate property and shrouded in secrecy. This growth in neural network applications is in part due to the availability of an increasingly wide array of dedicated neural network hardware, in the form of accelerator cards for PC's and workstations. Complementing the hardware are scores of commercial software packages (Rumelhart et al, 1993).

This study uses backpropagation artificial neural networks to examine whether they are capable of adequately capturing software development complexities in their weight space, to enable them to make accurate estimates.

RESEARCH METHODOLOGY

Data collection

For this study development data from 15 commercial systems, developed by information systems professionals, was recorded. The size of the systems was measured in unadjusted function points as it appears to be a more consistent measure than source lines of code (Jeffery & Low 1990; Kemerer, 1987; Low & Jeffery, 1990).

Development effort was measured in development hours. All activities, starting from the specification stage and through to that stage where the product is ready to be delivered to the end-user are included in the development time. The time spent on documentation is therefore also included. Excluded is the time required for the formal user acceptance tests, as well as end-user training. The definition of a development hour is the actual time spent on the project, and also includes all time spent on attending meetings directly related to the software development, but excludes times such as public holidays, leave, illness, and development staff training.

The development attributes which were included in the study were selected after reviewing some previously published research (Albrecht & Gaffney, 1983; Boehm, 1981; Conte et al, 1986; Jeffery, 1987; Jones, 1986; Vessey, 1986). The selection was refined by interviewing the information systems managers of 10 commercial organisations to establish which factors they with their development experience considered to have a significant impact on development productivity.

Precise descriptions of the various development attributes were developed, to attempt to ensure data consistency across the data set (Wittig, 1991). No interrater reliability study was conducted (Kemerer, 1993), as some of the systems are very large and this would have been costly. In the determination of the average size of the development team both analysts and programmers are included in the calculation, as well as project managers and program librarians directly involved with the development.

Research data

The systems ranged in size from a small 29 function point system to a system of 4669 function points. If uncommented 4GL source lines of code (SLOC) is used as a size measure, the systems ranged in size from 600 to 571 000 SLOC. The development effort required to develop these systems ranged from 40 to 81 270 hours. A system of less than 300 function points would be considered a small system, while a medium system would be between 300 and 800 function points, a large system between 800 and 1000 function points, and a very large system would be greater than 1000 function points (Dreger, 1989). A summary of these details is given in Table 1.

The size unit of unadjusted function points (UFP) is used. There has been some criticism regarding the choice and weighting of the function point adjustment factors (Jones, 1991; Kitchenham, 1992; Symons, 1991), and as it is easy to include these into the neural network model, the technical complexity adjustment to generate adjusted function points was not made. For this study software development productivity is defined in the economic sense (Jones, 1986) and is expressed as the amount of output produced per unit of input, and the unit used is unadjusted function points per development hour.

GL System	Dev Hours	Lines Code	Size UFP	Productivity UFP/Hr
1	5 027	88 325	1 842	0.37
2	1 680	44 204	905	0.54
3	13 300	170 557	4 191	0.32
4	98	1 535	208	2.12
5	588	2 019	342	0.58
6	450	1 600	164	0.36
7	40	1 400	29	0.73
8	81 270	298 843	4 113	0.05
9	35 000	562 500	3 486	0.10
10	240	600	286	1.19
11	100	3 241	214	2.14
12	4 864	25 000	2 758	0.57
13	1 200	30 000	1 913	1.59
14	2 500	571 000	4 669	1.87
15	1 400	–	850	0.61

Table 1 Data Set

An examination of the productivity range indicates that this varies from 0.05 to 2.14 function points per hour. Data with such a large size and productivity range, with significantly different development attributes, and which inherently contains a lot of noise, further complicates effort estimation. To convert the size of a system measured in function points to a development effort estimate, a productivity factor has to be applied. Productivity varies not only by programmer, but also by project size (Dreger 1989). This implies that the effect of project size on productivity has to be quantified to enable an accurate estimate of development effort to be made (Finnie and Wittig 1993). With a limited sample size multiple regression analysis proved difficult to derive accurate effort estimates, and attempts have been made using the Analytic Hierarchy Process (Finnie et al 1993) to prioritise the effect of the various development factors.

Neural network model

For this study, backpropagation artificial neural network models were used. Backpropagation networks are the most generalised neural networks currently in use (Nelson & Illingworth, 1991) and this approach was chosen in preference to Hopfield and Kohonen networks. Nelson and Illingworth (1991) discuss some of the many networks which have been developed and give guidelines for possible applications. As software development estimation is not a time series problem, approaches such as finite impulse response (FIR) and recurrent networks were not considered.
The backpropagation network requires data from which to learn. To learn the network calculates the error, which is the difference between the desired response and the actual response, and a portion of it is propagated backward through the network. At each neuron in the network the error is used to adjust weights and threshold values of the neuron, so that at the next epoch the error in the network response will be less for the same inputs. This corrective procedure is called backpropagation and is applied continuously for each set of inputs or training data. The training data should consist of as much relevant data as possible. In practice one does not usually have the luxury of a perfect training data set. With limited data one has to consider the trade-off between having as large a training set as possible and still leaving sufficient data points to test and validate the model.
For this project the data were divided into three sets. The training set comprised ten developments, the test set three, and the validation set two. The data for each category were randomly chosen, except that the data in the test and validation sets was not allowed to be larger or smaller than the largest and smallest developments respectively in the training set. This was done so that predictions were not made outside the data range on which the network had been trained.
The inputs were unadjusted function points, average development team size, systems analyst capability, systems analyst experience, the level of requirements volatility, the level of processing

complexity, whether reusable code was developed or not, the required processing reliability, and the capability and the experience of the programmer team. All the inputs except for the average development team size and the unadjusted function points reflecting the system size, were converted to a binary notation. The target against which the network was trained was the development hours of the systems in the training set. The accuracy of the development effort estimate was taken as the Root Mean Square Error (RMSE), proportionate to the size range of the systems in the data set.
To avoid over-training the network, the dangers of which are discussed later, and to be able to monitor the generalisation capability of the network, the training error and the prediction error were recorded at 100 epoch intervals. As the initial randomly generated starting network weights affect network performance these were saved, and the network was then re-run using the same starting weights, and stopped it when it had reached its minimum prediction error.

ANALYSIS OF DATA

Network models were developed with various combinations of inputs selected from the attributes mentioned above. The results were not particularly encouraging and the prediction errors were erratic and not satisfactory. An examination of the results showed that the network appeared to consistently overestimate the size of the very small systems, as well as to consistently underestimate the size of the very large systems. For the remaining systems, the estimated development effort was more accurate.

Network parameters

To try and improve the network performance, the learning rate and momentum were varied, as was the network architecture. Models with one through to six hidden layers were developed. Consistently the models with just a single hidden layer performed better, while the models with multiple hidden layers in many instances did not converge. Various activation functions were tried, and the popular sigmoid function consistently gave the best results.
To solve the problem of the network not training and predicting well on such a large target range, the natural logarithm of the development hours was used. This compressed the range and improved the network performance.
There is no clearly defined theory which allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behaviour of almost all networks (Schoneburg, 1990). It is through a process of trial and error and experience that settings are selected which will result in a reduced average prediction error. The settings of the learning rate and momentum control the way in which the error is used to correct the weights in the neural network for each training case. When the learning rate is set to high values (close to 1) there is the possibility of unstable behaviour, as evidenced by widely varying average error values. When the learning rate is set lower, the possibility of unstable behaviour is reduced, but training times are increased and there is a greater probability of getting stuck in local error minima. The higher the momentum, the larger the percentage of previous errors that is applied to the weight adjustment in each training case. For example, when the momentum is set at 0.5, then 50 percent of the weight adjustment will be due to the current error and 50 percent will be the weight adjustment applied in the previous case.
For this set of data a learning rate of 0.1 and a momentum of 0.7 gave good results. A neural network architecture of a single hidden layer using a sigmoid activation function tended to result in the lowest average prediction error. The best results were obtained with a 23-4-1 architecture, and Table 2 shows the results up to 2000 epochs. The average training error is reduced steadily as the network trains, as is the prediction error. For this network the lowest average prediction error was obtained at about 2000 iterations. With further training the training error is further reduced, but the network does not generalise well, and from this point the average prediction error increases. The reason for this is that the network tends to curve-fit the training data, giving a low average training error, but this then leads to poor generalisation.

Training Results		Prediction Results	
Iterations	Average Err	Iterations	Average Err
100	0.2748	100	0.1548
200	0.2060	200	0.1429
300	0.1504	300	0.1422
400	0.1210	400	0.1413
500	0.1063	500	0.1374
600	0.0971	600	0.1307
700	0.0892	700	0.1205
800	0.0820	800	0.1091
900	0.0757	900	0.0967
1000	0.0709	1000	0.0865
1100	0.0673	1100	0.0788
1200	0.0646	1200	0.0692
1300	0.0626	1300	0.0589
1400	0.0609	1400	0.0486
1500	0.0598	1500	0.0386
1600	0.0585	1600	0.0315
1700	0.0576	1700	0.0238
1800	0.0569	1800	0.0168
1900	0.0562	1900	0.0129
2000	0.0556	2000	0.0126

Table 2 Training and Prediction Errors

Network performance

A larger data set would have been preferable, but this type of information is difficult to gather. Other studies such as Kemerer (1987) (15 cases) and (1993) (27 cases) are also limited by data availability. A result of a small data set was that the network performance was significantly influenced by the initialisation weights which are randomly generated. Some sets of initialisation weights resulted in better convergence and a reduced average prediction error. As there is currently no known theory on the allocation of starting weights to optimise the network performance, and these are generated and allocated randomly, it meant that from repeated trials the weight set which resulted in the lowest average prediction error was selected.

The results of the network which gave the lowest average prediction error are shown in Table 3. The output has been normalised by converting it back from the natural logarithm and then rounding it. Both the actual system size and that predicted by the artificial neural network are shown. The first ten data sets comprise the training set. The next three comprise the test set, and the final two the validation set.

The error in the training set is not important. As mentioned above, by training the network further, this error could be further reduced, but this would have degraded the model's generalisation ability, resulting in an increased prediction error.

Examining the results in Table 3 reveals that neither the very smallest, nor the largest systems were included in the testing and validation set, as this would have meant that they would have been excluded from the training set, and would have resulted in having to predict outside this range. Training was stopped when the prediction error was at its lowest. The prediction capability of a network is judged by the test and validation sets. The three systems in the test set took 100, 1200 and 4864 person-hours to develop. The model predicted 89, 1184 and 4429 for the three systems respectively, resulting in error predictions of 11.1%, 1.3% and 9.0%.\

4GL System	Actual Development Hours	Estimated Development Hours	Percentage Error	Category
1	5 027	5 015	-0.24%	Training
2	1 680	1 652	-1.68%	Training
3	13 300	13 085	-1.62%	Training
4	98	269	174.41%	Training
5	588	377	-35.89%	Training
6	450	262	-41.69%	Training
7	40	62	55.42%	Training
8	81 270	62 236	-23.42%	Training
9	35 000	34 832	-0.48%	Training
10	240	240	0.11%	Training
11	100	89	-11.08%	Test
12	4 864	4 429	-8.95%	Test
13	1 200	1 184	-1.30%	Test
14	2 500	2 144	-14.24%	Validation
15	1 400	1 396	-0.27%	Validation

Table 3 Prediction Results

The two systems in the validation set required 1400 and 2500 person hours to develop. The prediction for the smaller system was 1396 person-hours, which is within 1% of the actual development time. For the other system the error was 14.2%.

DISCUSSION

Within the limited data set, backpropagation artificial neural networks appear to indicate the potential to be developed into good software size estimation models. In examining the model's performance the following factors need to be considered:

1. These systems were developed in an uncontrolled and natural (not artificial) environment. Only the data was gathered and there was no control over the development environment.
2. The software development environment is complex, with many and often interrelated factors affecting development effort. Currently available size estimation models have on occasions not performed well when applied to systems which were developed outside of a strictly controlled environment (Ferens & Gurner, 1992; Kemerer, 1987; Mukhopadhyay et al, 1992).
3. The range of the system sizes is large, as is the variation in productivity. Despite these difficulties the models performed well in terms of current level of prediction accuracy (Ferens & Gurner, 1992; Kemerer, 1987; Mukhopadhyay et al, 1992)
4. The research experience with the small data set highlighted the importance of the initial weights allocated to the network weight space. This makes the model development more difficult. Additional data, which is difficult to obtain in large sets, should reduce the influence of the starting weights and make training networks more stable (Lendaris, 1993).
5. The model is not difficult to develop and has the flexibility of being able to incorporate additional attributes as input if special circumstances warrant their inclusion.

FURTHER RESEARCH

The model has produced good results, as noted above, in being able to predict on average within 10 percent of the actual software development effort in the data set to which it was applied. Neural networks have the ability to capture knowledge of the complex interrelationships in their weight matrix to enable them to make predictions. It would be a distinct advantage if the model was also able to provide explanation to clarify the results of the reasoning process. Even if the weight matrix is accessible, it is difficult to interpret, since the user has no way to 'decompile' the weights.
Neural networks have no explicit, declarative knowledge structure which allows the representation of explanation structures, such as reasoning paths, and explanation of expectation failures. With a small training set as was used in this study it is not possible to have full confidence in training. It would

require a large set of training examples (of a few thousand instances for example) to allow the network to learn a 'complete' domain theory (Diederich, 1992).
Several proposals have been made for explanation components in plain neural networks as well as structured connectionist systems (Diederich, 1992). Further research will be conducted to study causality and explanation in neural network models. In this regard Connectionist Semantic Networks (Diederich, 1992) and Cascade Correlation Networks (Hoehfeld & Fahlman, 1991) are of interest.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their considered comments and helpful suggestions.

REFERENCES

Albrecht, A.J. & Gaffney, J.E. (1983) "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", **IEEE Transactions on Software Engineering**, Vol 9, No 6, pp 639–648.

Banker, R.D. Datar, S.M. & Kemerer, C.F. (1991) "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects", **Management Science**, Vol 37, No 1, pp 1–18.

Boehm, B.W. (1981) **Software Engineering Economics**, Englewood Cliffs, NJ: Prentice Hall.

Conte, S.D. Dunsmore, H.E. & Shen, V.Y. (1986) **Software Engineering Metrics and Models**, Menlo Park: Benjamin/Cummings.

de Marco, T. (1982) **Controlling Software Projects: management, measurement and estimation**, New York, NY: Yourdon Press.

Diederich, J. (1992) "Explanation and Artificial Neural Networks", **International Journal Man-Machine Studies**, Vol 37, pp 335–355.

Dreger, J.B. (1989) **Function Point Analysis**, Englewood Cliffs, NJ: Prentice Hall.

Ferens, D.V. & Gurner R.B. (1992) "An Evaluation of Three Function Point Models for the Estimation of Software Effort", **IEEE National Aerospace and Electronics Conference–NAECON 1992**, Vol 2, pp 625–642.

Finnie, G.R. & Wittig, G.E. (1993) "Effect of System and Team Size on 4GL Software Development Productivity", **South African Computer Journal**, to be published 1994.

Finnie, G.R. Wittig, G.E. & Petkov, D.I. (1993) "Prioritising Software Development Productivity Factors Using the Analytic Hierarchy Process", **Journal of Systems Software**, Vol 22, No 2, pp 129–139.

Hoehfeld, M. & Fahlman, S.E. (1991), "Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm", **Carnegie Mellon University Report**, CMU-CS-91-130.

Jeffery, D.R. (1987) "A Software Development Productivity Model for MIS Environments", **Journal of Systems Software,** Vol 7, pp 115–125.

Jeffery, D.R. & G.C. Low, (1990) "Calibrating Estimation Tools for Software Development, July, **Software Engineering Journal,** pp 215–221.

Jones, C. (1986) **Programming Productivity**, New York: NY: McGraw-Hill.

Jones, C. (1991) **Applied Software Measurement**, New York: NY, McGraw-Hill.

Kemerer, C.F. (1987) "An Empirical Validation of Software Cost Estimation Models", **Communications of the ACM**, Vol 30, No 5, pp 416–429.

Kemerer, C.F. (1993) "Reliability of Function Points Measurement", **Communications of the ACM**, Vol 36, No 2, pp 85–97.

Lendaris G.G. (1993) Professor of System Science and Electrical Engineering, Portland State University, **personal communication** 2 December 1993.

Low G.C. & Jeffery, D.R. (1990) "Function Points in the Estimation and Evaluation of the Software Process", **IEEE Transactions on Software Engineering,** Vol 1, No 1, pp 64–71.

Kitchenham, B.A. (1992) "Empirical Studies of Assumptions that Underlie Software Cost-Estimation Models", **Information and Software Technology**, Vol 34, No 4, pp 211–218.

Martin, J. (1982) **Application Development Without Programmers**, Englewood Cliffs, NJ: Prentice Hall.

Mukhopadhyay, T. Vicinanza, S.S. & Prietula, M.J. (1992) "Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation", **MIS Quarterly**, Vol 16, No 2, pp 155–171.

Nelson, M.M. & Illingworth, W.T. (1991) **A Practical Guide to Neural Nets**, Reading, Addison-Wesley.

Rudolph, E.E. (1986) "Productivity in Computer Application Development", **Auckland University Research Report**, Auckland University, Auckland, 1986.

Rumelhart, D.E. Widrow, B & Lehr, M. (1993) "Applications of Neural Networks in Industry, Business and Science", *Stanford University, unpublished.*

Schoneburg, E. (1990) "Stock Prediction Using Neural Networks: A Project Report", **Neurocomputing**, Vol 2, No 1, pp 17–27.

Vessey, I. (1986) "On Program Development Effort and Productivity", **Information and Management**, Vol 10, pp 255–266.

White, H. (1988) "Economic Prediction Using Neural Networks: The case of IBM Daily Stock Returns", **IEEE Conference on Neural Networks**, Vol 2.

Wittig, G.E. (1991) "An Analysis of 4GL Software Development Productivity", Masters Thesis, University of Natal.