ELSEVIER

Information and Software Technology 44 (2002) 911–922

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Comparison of artificial neural network and regression models for estimating software development effort

Abbas Heiat*

Management Information Systems, Montana State University—Billings, 1500 North 30th Street, Billings, MT 59101, USA

Received 26 October 2001; revised 27 December 2001; accepted 20 July 2002

Abstract

Estimating the amount of effort required for developing an information system is an important project management concern. In recent years, a number of studies have used neural networks in various stages of software development. This study compares the prediction performance of multilayer perceptron and radial basis function neural networks to that of regression analysis. The results of the study indicate that when a combined third generation and fourth generation languages data set were used, the neural network produced improved performance over conventional regression analysis in terms of mean absolute percentage error.

Keywords: Artificial neural network; Regression models; Software development effort

1. Introduction and background

Estimating the effort required to develop an information system is an important project management concern. As a result, a number of models have appeared in the literature over the past three decades. In a recent survey, Boehm et al., have provided an overview of currently available estimation models [8]. They classify estimation models into (1) model-based techniques, (2) expertise-based techniques, (3) learning-oriented techniques, and (4) dynamic-based techniques. Most software development effort estimators are model-based and parametric. Examples of model-based techniques include COCOMO [9], function points (FP) [2,3] and SLIM [13–16,21,39,42].

Constructive cost model (COCOMO) is based on a regression analysis of historical data from 63 completed projects. The basic model has the following form:

$$\mathrm{EF} = c(\mathrm{LOC})^k$$

where EF is the number of staff-months or hours required; c, a constant with an estimated value of 3.2; LOC, the lines of code (LOC) delivered in thousands and k, a constant with an estimated value of 1.05.

There are two steps in effort estimation using COCOMO. First, the number of LOC required for the information system is calculated. Second, the total effort is estimated by using $\mathrm{EF} = 3.2(\mathrm{LOC})^{1.05}$ equation and inputing calculated LOC in the equation [9]. The effort estimated by the COCOMO equation can be modified using considerations other than program size. The intermediate COCOMO model estimates effort as a function of line of code and 15 cost drivers [39]. The advanced COCOMO model uses LOC and a set of cost drivers weighted according to each phase of software development life cycle [18].

To address the issues emerging due to use of new software development methodologies like object-oriented programming, commercial-off-the-shelf application and CASE, Boehm and his collaborators started COCOMO II research effort in 1994 [9]. COCOMO II includes application composition, early design, and post architecture models. The application composition uses object points. Early design uses FP or LOC plus a set of scale factors and effort multipliers. Post architecture model is used when detailed information about the software development project is available [9].

An alternative method for estimating systems development effort was developed by Albrecht and Jones [1,12,31,36,41,45]. Albrecht introduced the concept of FP to measure the functional requirements of a proposed system. In FP modeling, the size of the system is determined by first

* Tel.: +1-406-657-1627; fax: +1-406-657-2223.
E-mail address: aheiat@msubillings.edu (A. Heiat).

0950-5849/02/$ - see front matter
PII: S0950-5849(02)00128-3

identifying the type of each required function in terms of inputs, outputs, inquiries, internal files, and external interface files. To calculate the value of FP for each category, the number of functions in each category is multiplied by the appropriate complexity weight. The total systems effort is then calculated by multiplying the sum of FP for all categories by the technical complexity factor (TCF). The TCF is determined by assigning values to 14 influencing project factors and totaling them. Albrecht argued that FP model makes intuitive sense to users and it would be easier for project managers to estimate the required systems effort based on either the user requirements specification or logical design specification. Another advantage of the FP model is that it does not depend on a particular language. Therefore, project managers using the FP model would avoid the difficulties involved in adjusting the LOC counts for information systems developed in different languages. Reviews and critiques of the FP model appear in Refs. [27,28,53,50], Low et al. (1993) and Ref. [52].

Putnam developed the software life cycle model (SLIM) which estimates development effort using LOC as the primary input. It uses Rayleigh distribution for resources consumed by the project [42]. The Rayleigh curve used to define the distribution of effort is modeled by the differential equation:

$$\mathrm{d}y/\mathrm{d}t = 2Kat\exp(-at^2)$$

where $\mathrm{d}y/\mathrm{d}t$ is the staffing rate, K represents the total manpower required, and a is a shape parameter. In order to estimate project effort (K) or development time (t_d) two equations have been introduced and can be derived after several steps.

$$t_d = [(S)/(D_0C^3)]^{1/7}$$

$$K = (S/C)^{9/7}(D_0)^{4/7}$$

S is system size measured in thousand LOC, D_0 is the manpower acceleration, and C is called the technology factor. To apply the Putnam's model it is necessary to determine the C, S and D_0 parameters up-front [40].

Artificial neural network (ANN) is the most commonly used learning-oriented approach for estimating software development effort [22]. Neural computing requires a number of elementary processing units, called neurons, to be connected together into a neural network. In most common networks, neurons are arranged in layers with the input data fed to the network at the input layer. The data then passes through the network to the output layer to provide the solution or answer. Each neuron within the network is usually a simple processing unit which takes one or more inputs and produces an output. At each neuron, every input has an associated weight which modifies the strength of each input connected to that neuron, as illustrated in Fig. 1.

During the training process, as each example is presented to the network, the values of the 'weights' are adjusted to bring the output of the whole network closer to the desired output. The training phase is concluded when the network provides sufficiently accurate answers to all the training problems. The number of neurons in a neural network can range from tens to thousands. The way in which the neurons are organized is called the network topology and the most popular topology in use today is shown in Fig. 1. This consists of three layers of neurons in which the output of each neuron is connected to all the neurons in the next layer. This is known as a feed-forward network. Data flows into the network through the input layer, passes through one or more intermediate hidden layers, and finally flows out of the network through the output layer. A single neuron has a very limited capability as a device for solving problems. However, when a number of neurons are connected together to form a complex system, it is possible for the network to be trained to offer a meaningful solution.

There are many techniques for training a neural network. The two main techniques employed by neural networks are known as supervised learning and unsupervised learning. In unsupervised learning, the neural network requires no initial information regarding the correct classification of the data it is presented with. The neural network employing unsupervised learning is able to analyze a multidimensional data set

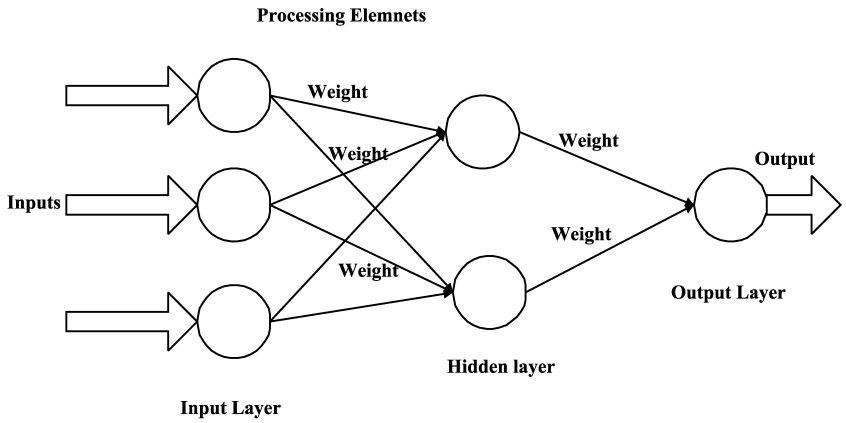

Fig. 1. Architecture of a neural network.

in order to discover the natural clusters and subclusters that exist within that data. Neural networks using this technique are able to identify their own classification schemes based upon the structure of the data provided, thus reducing its dimensionality. Unsupervised pattern recognition is therefore sometimes called cluster analysis [47]. Supervised training works in much the same way as a human learns new skills, by showing the network a series of examples. The most common supervised training algorithm is known as back propagation. An overview of the back propagation training method is as follows:

1. A set of examples for training the network is assembled. Each case consists of inputs into the network and the corresponding solution which represents the desired output from the network.
2. The input data is entered into the network via the input layer.
3. Each neuron in the network processes the input data, with the resultant values steadily going through the network, layer by layer, until a result is generated by the output layer.
4. The actual output of the network is compared to expected output for that particular input. This results in an error value which represents the discrepancy between given input an expected output. On the basis of this error value all of the connection weights in the network are gradually adjusted, working backwards from the output layer, through the hidden layer, and to the input layer, until the correct output is produced. Fine tuning the weights in this way has the effect of teaching the network how to produce the correct output for a particular input.

Steps 2–4 above are repeated, potentially involving many of examples. As the training process proceeds, the connections which lead to the correct answers are strengthened, and the incorrect connections are weakened. The training process is complete when the network produces the correct output for every input case. The network's weights are saved in their trained states and the network is then ready for use. New input data are presented to the network and the network will determine the appropriate output on the basis of its training. The answer given by a network may not be absolutely correct but rather the optimum or best answer.

2. Performance evaluation of estimation models

A key factor in selecting a software estimation model is the accuracy of its predictions. According to Leung and Fan, many studies have attempted to evaluate the performance of software effort estimation models. However, many of them were found to be not very accurate [30]. In a study by Kemerer using COCOMO, FP, SLIM, and Estimacs, most models showed large estimation errors, ranging from a MAPE of 57 to 800% [28].

Vicinanza et al. using Kemerer data set, found that experts outperformed the original model-based techniques in estimating the software development effort. However, the MAPE ranged from 32 to 1107% [52]. Ferens and Gumer evaluated three models using 22 projects from Albrecht's database and 14 projects from Kemerer's data set. Reported MAPE ranged from 46 to 105% [23]. A study by Miyazaki et al. using COCOMO also found high MAPE error rates, averaging 166% [35].

In two studies, Jeffery and Low investigated the impact of the model calibration. In the first study without model calibration, the MAPE ranged from 43 to 105% [25]. In the second study, they recalibrated their models for 64 projects from a single organization to the local environment. They reported substantial improvement in effort estimate with a MAPE of 12% [26]. However, in an extensive study from 1995 to 1997, graduate students at Air Force Institute of Technology calibrated nine parametric software effort estimation models. Results indicated that the calibration does not always improve estimation performance of parametric models [19].

In the last two decades, ANN have been used for predictions in diverse applications. In general, results have been superior to conventional methods [4,6,7]. In recent years, a number of studies have used neural networks in various stages of software development. Hakkarainen et al. estimated software size by training an ANN. They used structured specification descriptions as input and demarco function bang, Albrecht's FP and Symon's mark II FP size metrics as output. The results of their study indicated that ANN could be used successfully to estimate software size [24]. Srinivasan and Fisher compared two approaches (1) a back propagation neural network and (2) regression trees, using Boehm's historical database. Their experiments indicated that neural network and regression trees are competitive with model-based approaches [49]. Finnie and Wittig applied ANN and case-based reasoning (CBR) to estimate software development effort [20]. They used a data set from the Australian Software Metrics Association. ANN was able to estimate software development effort within 25% of the actual effort in more than 75% of the cases, and with a MAPE of less than 25%. Mair et al. used 67 software projects derived from a Canadian software house to evaluate prediction performances of regression, rule induction (RI), CBR and ANN techniques [32]. The results of the study showed considerable variation between the four models. MAPE for RI ranged from 86 to 140%. MAPE for regression ranged from 38 to 100%. MAPE for CBR ranged from 43 to 80% and for ANN ranged from 21 to 66%. MAPE results suggest that ANN seem to be the most accurate and RI is the least accurate technique [32]. Shukla conducted a large number of simulation experiments using genetically trained neural networks. He used a merged database comprising 63 projects, and Kemerer database comprising 15 projects. The

results indicated a significant estimation improvement over quick propagation network and regression trees approaches. Shukla concluded that there is still a need to apply neural networks to diverse projects with wide range of attributes because it is "unclear which techniques are most valuable for a given problem…, experimental comparison using rigorous evaluation methods is necessary" [46].

According to Briand et al. studies of performance evaluation of estimation models are rarely replicated. They believe replication, particularly when existing results are inconclusive, is critical for validation and generalization of estimation models [11].

The purpose of this study is twofold: (1) it reports the results of using a third generation programming language (3GL) database as compared to a combined 3GL and fourth generation programming languages (4GL) database, (2) it compares conventional regression analysis with a neural network model using both the third generation programming language data set and the combined bi-language data set.

The remainder of this paper is organized as follows. Section 3 presents a brief description of the multilayer perceptron (MLP) and radial basis function (RBF) ANN models used in this study. Section 4 describes the data collection method and previous related studies. Section 5 explains analysis methodology of experimental studies reported in this paper that leads into Section 6, estimation results and discussion. Finally, Section 7 offers conclusions and a recommendation for future research.

3. Multilayer perceptron and radial basis function neural networks

The MLP is one of the most widely implemented neural network topologies. In terms of mapping abilities, the MLP is believed to be capable of approximating arbitrary functions. This has been important in the study of non-linear dynamics, and other function mapping problems. MLPs are normally trained with the back propagation algorithm. Two important characteristics of the MLP are:

1. Its smooth non-linear processing elements (PEs). The logistic function and the hyperbolic tangent are the most widely used.
2. Their massive interconnectivity, i.e. any element of a given layer feeds all the elements of the next layer.

The MLP is trained with error correction learning, which means that the desired response for the system must be known. Back propagation computes the sensitivity of a cost function with respect to each weight in the network, and updates each weight proportional to the sensitivity [38].

The RBF network is a popular alternative to the MLP which can offer advantages over the MLP in some applications. An RBF network can be easier to train than an MLP network. The RBF network has a similar form to the MLP in that it is a multilayer, feed-forward network. However, unlike the MLP, the hidden units in the RBF are different from the units in the input and output layers. They contain the RBF, a statistical transformation based on a Gaussian distribution from which the neural network's name is derived. Like MLP neural networks, RBF networks are suited to applications such as pattern discrimination and classification, pattern recognition, interpolation, prediction and forecasting. In the hidden layer of an RBF, each hidden unit takes as its input all the outputs of the input layer xi. The hidden unit contains a basis function, which has the parameters center and width. The center of the basis function is a vector of numbers, ci, of the same size as the inputs to the unit and there is normally a different center for each unit in the neural network. The first computation performed by the unit is to compute the radial distance, d, between the input vector xi and the center of the basis function, typically using Euclidean distance:

$$d = \mathrm{SQRT}((x^1 - c^1)^2 + (x^2 - c^2)^2 + \cdots (x^n - c^n)^2)$$

The unit output, a, is then computed by applying the basis function B to this distance divided by the width w: $a = B(d/w)$

As you can see from Fig. 2, the basis function is a curve, typically a Gaussian function, which has a peak at zero distance and which falls to smaller values as the distance from the center increases. As a result, the unit gives an output of one when the input is centered but which reduces as the input becomes more distant from the center [10].

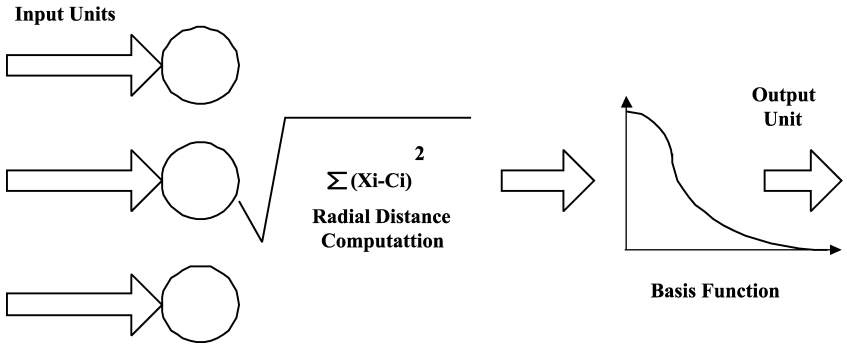

Fig. 2. Radial basis function neural network.

The output layer of an RBF neural network is essentially the same as for the MLP. Normally it has a linear activation function, making it possible to calculate the weights for those units directly. However, if the output units have non-linear activation functions, then iterative training algorithms must be used [10].

4. Data and previous related studies

The data used in this study came from three sources: The IBM DP Services Organization comprising 24 projects, Kemerer data set comprising 15 projects, and Hallmark data set comprising 28 projects [2,3,28] (Drummond, 1985). The IBM and Kemerer projects are medium to large size and they are developed by using third generation languages. Hallmark projects are medium size and are developed by using a wide variety of fourth generation languages. Tables 1–3 contain background data on the projects. The man-month estimates were converted to man-hour to express estimates of effort in all three databases in the same unit of measurement. LOC are estimated and listed in thousand of lines.

The IBM DP Services database contains business applications, 18 of which were written in COBOL, four written in PL1, and two written in DMS language. Using IBM database, Albrecht and Gaffney computed FP counts and work-hours for each project. They used ordinary least-square (OLS) regression to estimate effort (E) as a function of FP:

$$E = 13.39 + 0.0545\text{FP} \qquad R^2 = 0.874$$

While explanatory power is relatively high at $R^2 = 0.874$, Matson et al. noted autocorrelation and non-normal residuals problems. They believe that relationship between effort and FP are not linear and a quadratic FP improves the autocorrelation and normality of residuals [33].

The source for the Kemerer database is a national

Table 1
IBM DP services data

Actual effort-hours (1000)	Function points	Lines of code (1000)
102.4	1750	130
105.2	1902	318
11.1	428	20
21.1	759	54
28.8	431	62
10	283	28
8	205	35
4.9	289	30
12.9	680	48
19	794	93
10.8	512	57
2.9	224	22
7.5	417	24
12	682	42
4.1	209	40
15.8	512	96
18.3	606	40
8.9	400	52
38.1	1235	94
61.2	1572	110
3.6	500	15
11.8	694	24
0.5	199	3
6.1	260	29

Table 2
Kemerer data

Actual effort-hours (1000)	Function points	Lines of code (1000)
43.62	1217.1	253.6
12.54	507.3	40.5
168.31	2306.8	450
13.21	788.5	214.4
51.12	1337.6	449.9
12.77	421.3	50
3.53	99.9	43
19.81	933	200
17.63	1592.9	289
10.94	240	39
39.32	1611	254.2
35.07	789	128.6
23.86	690	161.4
37.53	1347.5	164.8
10.62	1044.3	60.2

Table 3
Hallmark data

Actual effort-hours (1000)	Function points	Lines of code (1000)
0.59	73.00	5.00
0.70	121.00	11.93
0.86	111.00	9.83
0.88	55.00	6.93
0.92	94.00	12.00
0.96	59.00	9.38
1.18	72.00	26.82
1.18	144.00	17.74
1.19	67.00	12.71
1.32	87.00	35.28
1.55	320.00	25.64
1.75	86.00	10.30
1.91	77.00	22.98
2.18	108.00	35.50
2.26	148.00	26.93
2.26	174.00	44.00
2.37	341.00	17.09
2.44	684.00	19.25
3.95	697.00	70.39
4.02	507.00	106.00
4.18	170.00	35.47
4.66	314.00	49.52
5.01	293.00	66.00
6.84	434.00	64.18
7.57	738.00	28.00
11.42	1206.00	50.38
13.14	791.00	72.75
23.30	1284.00	126.33

software development consulting firm. The database includes a variety of software applications. The database contains medium to large size business applications, 12 of which written in COBOL, one in Bliss, and two in Natural. Using regression analysis, Kemerer evaluated two LOC-based (SLIM, COCOMO) models and FP model for software cost estimation. He reported the following results:

$$\text{Man-months} = 49.9 + 0.082(\text{SLIM}) \qquad R^2 = 0.878$$

$$\text{Man-months} = 27.7 + 0.155(\text{COCOMO-Basic})$$

$$R^2 = 0.68$$

$$\text{Man-months} = 62.1 + 0.123(\text{COCOMO-Intermediate})$$

$$R^2 = 0.599$$

$$\text{Man-months} = 66.8 + 0.118(\text{COCOMO-Detailed})$$

$$R^2 = 0.525$$

$$\text{Man-months} = -122 = 0.341(\text{Function Points})$$

$$R^2 = 0.553$$

While R^2 for SLIM model is relatively high, SLIM did not do well in terms of magnitude of relative error (MRE) test. Kemerer reported that the average percentage error was 772. All three COCOMO models did poorly in terms of R^2 and MRE test. The average error for all versions was 601%. FP model did poorly according to R^2. However, the average MRE was 102.74%, much better than the SLIM and COCOMO models [28].

LOC and FP data collected at hallmark came from projects developed and written in a non-procedural fourth generation languages (Drummond, 1985). To the best of my knowledge, this data set has not been used in any study.

5. Experimental studies

I conducted two experiments with regression analysis and neural network to compare estimation accuracy of software development effort by these two approaches. The projects from Kemerer and IBM data sets, which include third generation programming languages (3GL), were combined for the first experiment. Several researchers have concluded that the effort estimation models are generally valid within the organization and platform in which they were developed [5,34]. To examine this observation, for the second experiment, projects from Kemerer, IBM, and Hallmark data sets which include both third generation and fourth generation programming languages (4GL) were combined. This made it possible to compare regression analysis with neural network on 3GL platform and on combined 3GL and 4GL platforms.

The following regression model was selected for evaluating and testing the accuracy of the LOC, and FP:

$$\text{EFH} = c(\text{LOC/FP})^k$$

where EFH is the number of staff-hours required; c, a constant; LOC, the lines of code delivered excluding comments and utility software; FP, the computed function points, and K, a constant.

This author using the statistical package EView performed regression analysis. Equations are estimated by OLS estimators covering 32 cases for the first experiment and 60 cases for the second experiment. Forecasts of the dependent variable, EFH, were computed for each equation using test data covering seven cases.

Using NeuroSolutions software, a MLP neural network and a RBF network with one, two, and three hidden layers and randomly selected initial weights was created for this study. The size of the mean square error (MSE) can be used to determine how well the network output fits the desired output. The simulation runs terminates when the MSE drops below the specified threshold. The specified threshold for experiments were set to 0.01 which means that the simulations for finding the best network configuration would stop when average error or the difference between actual effort and estimated effort for the training set drops below 1%.

The size of the training set is of fundamental importance to the practical usefulness of the network. If the training patterns do not convey all the characteristics of the problem class, the mapping discovered during training only applies to the training set. Thus the performance in the test set will be much worse than the training set performance. Another aspect of proper training is related to the relation between training set size and number of weights in the ANN. If the number of training examples is smaller than the number of weights, one can expect that the network may 'hard code' the solution, i.e. it may allocate one weight to each training example. This will obviously produce poor generalization. NeuroSolutions recommends that the number of training examples be at least double the number of network weights. When there is a big discrepancy between the performance in the training set and test set, we can suspect deficient learning. Note that one can always expect a drop in performance from the training set to the test set. At our present stage of knowledge, establishing the size of a network is more efficiently done through experimentation. The issue is the following: the number of PEs in the hidden layer is associated with the mapping ability of the network. The larger the number, the more powerful the network is. However, if one continues to increase the network size, there is a point where the generalization gets worse. This is due to the fact that we may be over-fitting the training set, so when the network works with patterns that it has never seen before the response is unpredictable. The problem is to find the smallest number of degrees of freedom that achieves the required performance in the test set. One school of thought

recommends starting with small networks and increasing their size until the performance in the test set is appropriate. Some researchers propose a method of growing neural topologies that ensures a minimal number of weights, but the training can be fairly long. An alternate approach is to start with a larger network, and remove some of the weights. There are a few techniques, such as weight decay, that partially automate this idea. The weights are allowed to decay from iteration to iteration, so if a weight is small; its value will tend to zero and can be eliminated by the development life cycle. In this study I started with one hidden layer and then increased the size of the network to two and three hidden layers. The weight decay approach was used for all networks.

To compare and evaluate the accuracy of the estimators, the mean absolute percentage error (MAPE) of the forecasted values with respect to the actual amount of the development effort-hours (EFH) for each project was calculated:

$$\text{MAPE} = (\sum(|V_{\text{est}} - V_{\text{act}}|/V_{\text{act}}))/N \times 100$$

where V_{est} is the forecasted value of EFH and V_{act} is the actual EFH consumed by the project. N is the number of cases used in the test sample. Training set for neural network models and estimation set for regression analysis included 32 cases for the first experiment and 60 for the second experiment. Forecasts of the dependent variable, EFH, for regression and neural network models were computed using test data covering seven cases.

6. Results of the experimental studies

6.1. Experiment 1: 3GL platform using combined data from Kemerer and IBM

Regression estimation results are summarized in Table 4. Overall, the regression results are satisfactory, producing relatively high value of coefficient of determination (R^2). The t statistics indicate that all regressions coefficients are significant.

The critical values for Durbin–Watson (DW) test at 5% level of significance are: $d_L(0.05, 32, 1) = 1.37$ and $d_U(0.05, 32, 1) = 1.5$. Therefore, the hypothesis of no autocorrelation for the FP regression is accepted. The DW test for the LOC regression is inconclusive.

As mentioned earlier, MLP and RBF networks with one, two, and three hidden layers were developed in this study. The tolerance level for the MSE which indicates how well the network output fits the desired output was set to 0.01.

The best fit model may be determined by R^2 in regression analysis and MSE in neural network models. However, the consistency and accuracy of models should be assessed by using an out-of-sample validation data set that has not exerted any influence on the estimation or development of the models.

A summary of the MAPE results for regression analysis and neural network is shown in Table 5. The actual and the forecasted values for seven out-of-sample cases are shown in Table 5 and Figs. 3–6.

Regression analysis using LOC and RBF neural network with three hidden layers using LOC as input produced the best MAPE results. MAPE results for regression analysis and neural network are very close and almost identical for 3GL platform. The MAPE results of this experiment are clearly superior to the MAPE results (average 601% for LOC and average 102.745 for FP) of the experiments conducted by Kemerer. This may be mainly due to a larger data set that was used in this study.

Table 4
Estimated regression equations for combined data from IBM and Kemerer projects

(1)	Function points (FP)	$\log(\text{EFH}) = -6.08 + 1.3757 \log(\text{FP})(-6.79)(9.79)$ $R^2 = 0.76$ DW = 1.55
(2)	Lines of code (LOC)	$\log(\text{EFH}) = -1.2288 + 0.951 \log(\text{LOC})(-2.82)(9.13)$ $R^2 = 0.74$ DW = 1.37

Table 5
Mean absolute percentage error (MAPE) for combined data from Kemerer and IBM projects

Regression analysis			Neural network		
Actual effort	Estimated effort/FP	Estimated effort/LOC	Actual effort	Estimated effort/ FP/MLP	Estimated effort/ LOC/RBF
17.63	58.88	64.22	17.63	68.01	59.75
10.94	4.36	9.55	10.94	11.31	8.71
39.32	59.80	56.84	39.32	69.81	48.76
35.07	22.40	29.72	35.07	19.82	43.46
23.86	18.36	36.89	23.86	17.46	37.28
37.53	46.77	37.63	37.53	46.65	36.87
10.62	32.94	14.44	10.62	28.74	17.86
MAPE	91.30	61.10	MAPE	90.27	61.91

6.2. Experiment 2: combined 3GL and 4GL data from Kemerer, IBM, and Hallmark

Regression results for the second experiment are presented in Table 6. Values of coefficient of determination (R^2) are relatively high and the t statistics indicate that all regressions coefficients are significant.

The critical values for DW test at 5% level of significance are: $d_L(0.05, 60, 1) = 1.55$ and $d_U(0.05, 60, 1) = 1.62$. Therefore, we may reject the hypothesis of no autocorrelation for both FP and LOC regression. When disturbances exhibit autocorrelation, least-squares estimates of the regression coefficients are unbiased, but the estimated

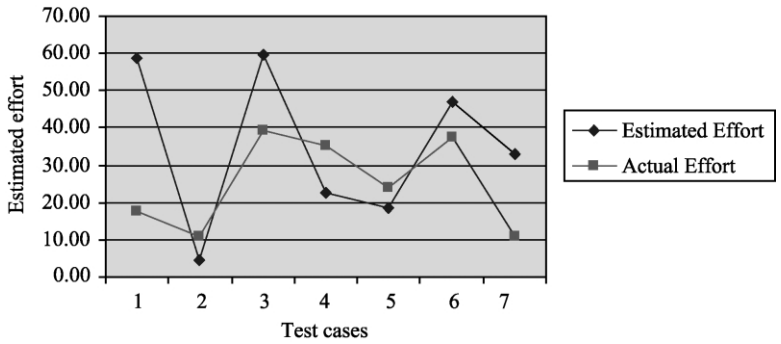

Fig. 3. Effort estimated by regression using function points and data from Kemerer and IBM projects.

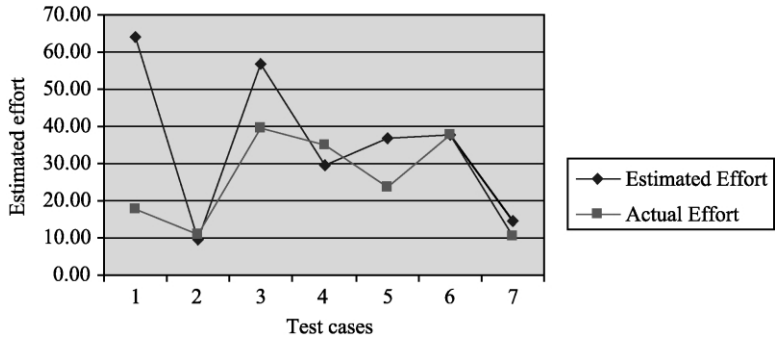

Fig. 4. Effort estimated by regression using lines of code and data from Kemerer and IBM projects.

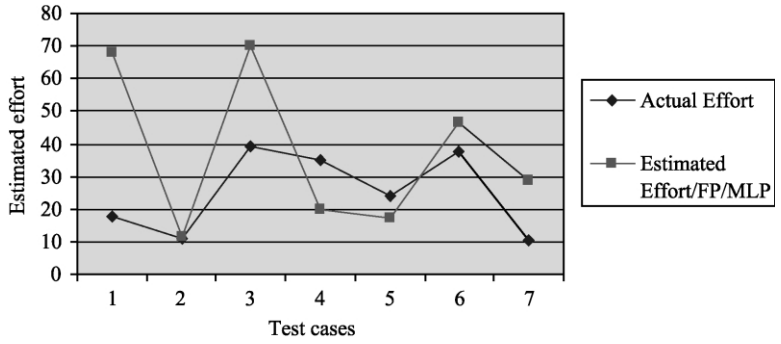

Fig. 5. Effort estimated by MLP neural network using function points and data from Kemerer and IBM projects.

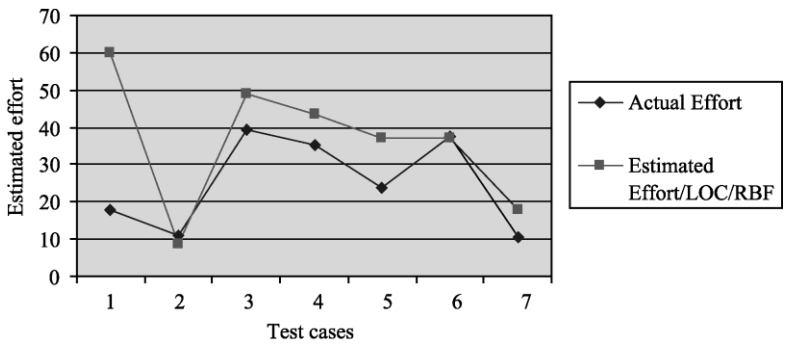

Fig. 6. Effort estimated by RBF neural network using lines of code and data from Kemerer and IBM projects.

Table 6
Estimated regression equations for combined data from all three projects

(1)	Function points (FP)	$\log(EFH) = -5.37 + 1.236 \log(FP)(-10.45)(14.4)$ $R^2 = 0.77$ DW = 1.35
(2)	Lines of code (LOC)	$\log(EFH) = -2.28 + 1.112 \log(LOC)(-6.68)(12.83)$ $R^2 = 0.73$ DW = 1

Table 7
Mean absolute percentage error (MAPE) for combined data from all three projects

Regression analysis			Neural network		
Actual effort	Estimated effort/FP	Estimated effort/LOC	Actual effort	Estimated effort/ FP/RBF	Estimated effort/ LOC/RBF
11.42	29.95	7.99	11.42	26.75	9.55
13.14	17.78	12.03	13.14	10.90	15.85
23.30	32.36	22.22	23.30	32.45	25.32
38.10	30.84	15.99	38.10	38.74	19.54
61.20	41.56	19.05	61.20	63.38	22.30
3.60	10.09	2.08	3.60	7.85	3.72
11.80	15.13	3.50	11.80	9.52	4.43
MAPE	70.86	40.37	MAPE	47.60	31.96

regression coefficients have larger variance than estimated coefficients with no autocorrelation. As a result, values of prediction errors by auto correlated regression equations will be larger than expected.

MAPE results for regression analysis and neural network is presented in Table 7. The actual and the forecasted values for seven out-of-sample cases are shown in Table 7 and Figs. 7–10.

The MAPE result s for RBF neural network using FP (47.6%) and RBF neural network, using LOC (31.96%) are superior to that of regression (70.86% for FP) and (40.37% for LOC) (Table 7). The critical value for a two-tailed test at 0.05 level of significance (2.447) is smaller than the obtained value of t (3.08) for MAPE values for regression/FP and neural network/FP models. In other words, the t-test indicates that a significant difference exists between regression and ANN approaches using FP. However, the critical value of t (2.447) is larger than the obtained value of t (1.34) for MAPE values for regression/LOC and neural network/LOC models. Therefore, the t-test indicates that there is no significant difference exists between regression and ANN approaches using LOC.

SubbaNarasimha et al. reported that ANN-based models perform better than regression techniques when sample distribution is not normal and data is skewed [48]. Probability plots are generally used to determine whether the distribution of a variable matches a given distribution. If the selected variable matches the test distribution, the points cluster around a straight line. Figs. 11 and 12 illustrate the normal probability plot of actual values of software development effort for the first and second experiment. Since the points are not clustered around the straight line for both data sets, sample distribution of development effort of both data sets is not normal. Therefore, non-normality of the sampling distributions cannot account for the superior MAPE results obtained for neural network in the second experiment.

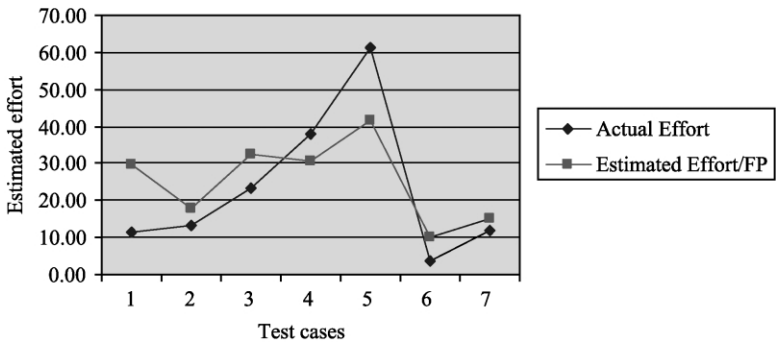

Fig. 7. Effort estimated by regression using function points and data from all three projects.

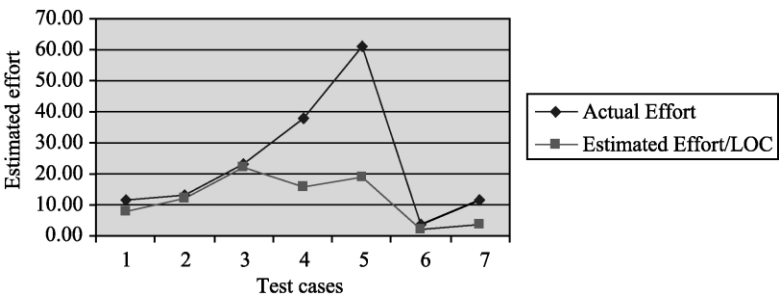

Fig. 8. Effort estimated by regression using lines of code and data from all three projects.

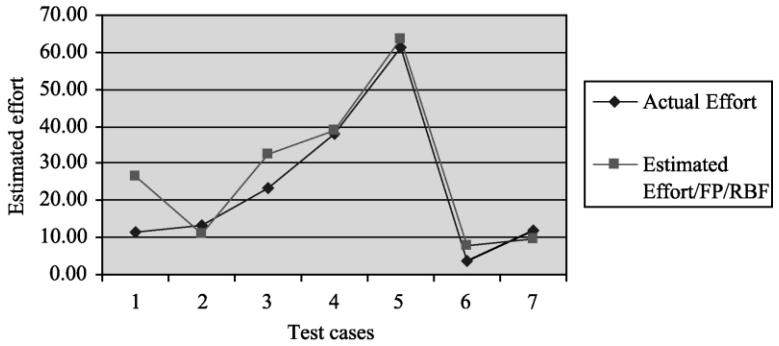

Fig. 9. Effort estimated by RBF neural network using function points and data from all three projects.

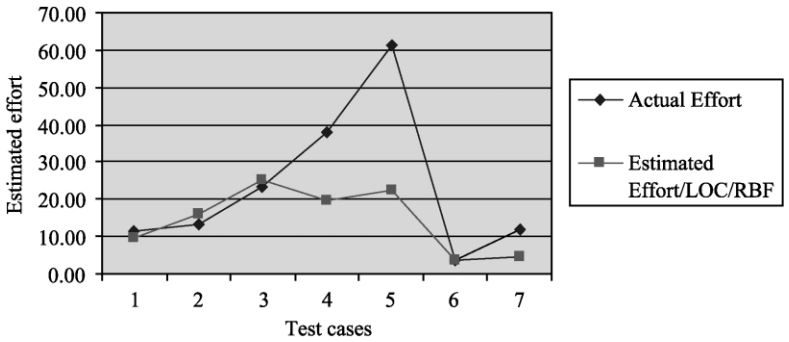

Fig. 10. Effort estimated by RBF neural network using lines of code and data from all three projects.

A second explanation has to do with the complexity of the system. Neural networks are usually efficient in predicting complex systems that demonstrate non-linear behavior. In the second experiment, I used a combination of third generation and fourth generation languages which makes the data set more complex. In addition, FP approach captures the complexity of software development effort better than LOC and does not depend on a particular language. In other words, superior performance of neural network in the second experiment and particularly when FP is used as an input may be due to complexity of the combined 3GL and 4GL data sets and language-independence property of the FP approach.

7. Concluding remarks

This article has compared the neural network estimation

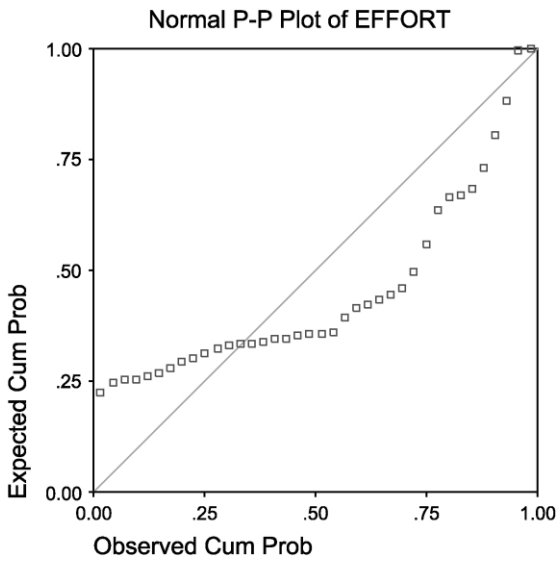

Fig. 11. Normality test for combined data from Kemere and IBM.

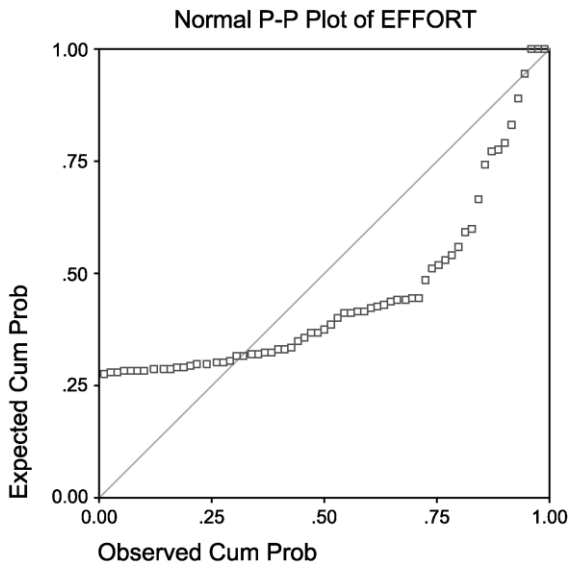

Fig. 12. Normality test for combined data from Kemerer, IBM, and Hallmark.

method to regression approach for software effort estimation. The results of this preliminary research indicate that neural network approach was competitive with regression when a third generation language data set was used in the study. However, in the second study when a combined third generation and fourth generation languages data set were used, the computed MAPE of the neural network model was significantly smaller than MAPE of regression model.

We should be aware of the following limitations of this study. First, although the size of data sets I used were larger than original Kemerer data set, further studies with larger and more diversified platforms are needed to verify the results of this study. Second, the system development process is a complex and multidimensional process. It is affected by a number of project attributes including the size of project, software development tools, analysts, and programmers' experience, and software development methodology. In this study, all of the above factors have been held constant. The data sets used varied only in size and complexity in terms of language platform. One possible extension of this research is to incorporate one or more productivity factors into the ANN models and determine their impact on the estimating software development effort.

References

[1] A.J. Albrecht, Measuring Application Development Productivity, Proceedings of the IBM Applications Development Symposium, GUIDE/SHARE, October, 1979, pp. 83–92.

[2] A.J. Albrecht, J. Gaffney, Software function, source lines of code, and development effort prediction: a software science validation, IEEE Transactions on Software Engineering SE-9 (6) (1983) 639–648.

[3] A.J. Albrecht, J. Gaffney, Software function, source lines of code, and development effort prediction, IEEE Transactions on Software Engineering 9 (6) (1983) 639–648.

[4] B. Barbro, Neural networks and genetic algorithms for bankruptcy predictions, Expert Systems with Application 11 (4) (1996) 407–413.

[5] J.W. Bailey, V.R. Basili, A Meta-Model for Software Development Resource Expenditure, Proceedings of the Fifth International Conference on Software Engineering, San Diego, CA, 1981.

[6] M.J.A. Berry, G. Linoff, Data Mining Techniques for Marketing Sales, and Customer Support, Wiley, New York, 1997.

[7] J.P. Bigus, Data Mining with Neural Networks, McGraw-Hill, New York, 1996.

[8] B. Barry, C. Abts, S. Chulani, Software development cost estimation approaches—a survey, Annals of Software Engineering February (2000).

[9] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[10] BrainStorm, Overview of Neural Computing, 2001, www.brainstorm.co.uk.

[11] L. Briand, T. Langley, I. Wieczorek, A replicated assessment and comparison of common software cost modeling techniques, International Software Engineering Network, Technical Report ISERN-99-15, 1999.

[12] C. Jones, Applied Software Measurement, McGraw-Hill, Englewood Cliffs, NJ, 1996.

[13] S.D. Conte, H.E. Dunsmore, V.Y. Shen, Software Engineering Metrics and Models, The Benjamin/Cummings Publishing Company, Inc, Menlo Park, CA, 1986, p. 214.

[14] V.P. Cot, B. Oligny, N. Rivard, Software metrics: an overview of recent results, The Journal of Systems and Software 8 (1988) 121–131.

[15] W.H. Delone, Determinants of success for computer usage in small business, MIS Quarterly 12 (1) (1988) 51–61.

[16] D.M. Tom, Controlling Software Projects, Yourdon Press, New York, 1982.

[18] N.E. Fenton, S.L. Pfleeger, Software Metrics: a Rigorous and Practical Approach, International Thomson Computer Press, London, UK, 1997.

[19] D.V. Ferens, D.S. Christensen, Calibrating software cost models to DOD databases—a review of 10 studies, Journal of Parametrics 14 (1999) 33–52.

[20] G.R. Finnie, G.E. Wittig, AI Tools for Software Development Effort Estimation, Software Engineering and Education and Practice Conference, IEEE Computer Society Press, Silver Spring, MD, 1996, pp. 346–353.

[21] J.R. Golden, J.R. Mueller, B. Anselm, Software cost estimating: craft or witchcraft, DATABASE (1981) 12–14.

[22] A. Gray, S. MacDonell, A comparison of techniques for developing predective models of software metrics, Information and Software Technology (1997) 39.

[23] D.V. Ferens, R.B. Gumer, An Evaluation for Three Function Point Models of Estimation of Software Effort, IEEE National Aerospace and Electronics Conference, 1992.

[24] J. Hakkarainen, P. Laamamen, R. Rask, Neural networks in specification level software size estimation, in: P.K. Simpson (Ed.), Neural Network Applications, IEEE Technology Update Series, 1993, pp. 887–895.

[25] D.R. Jeffery, G.C. Low, Calibrating estimation tools for software development, Software Engineering Journal (1990) 215–221.

[26] D.R. Jeffery, G.C. Low, M. Barnes, A comparison of function point counting techniques, IEEE Transactions on Software Engineering 19 (5) (1993) 529–532.

[27] C.F. Kemerer, Software Cost Estimation Models, Software Engineers Reference Handbook, Butterworth, Surrey, UK, 1991, Chapter 25.

[28] C.F. Kemerer, An empirical validation of software cost estimation models, Communications of the ACM 30 (1987) 416–429.

[29] B. Kitchenham, N.R. Taylor, Software cost models, ICL Technical Journal (1984) 73–102.

[30] H. Leung, Z. Fan, Software Cost Estimation, Handbook of Software Engineering, 2002.

[31] G.C. Low, D.R. Jeffery, Function points in the estimation and evaluation of the software process, IEEE Transactions of Software Engineering January (1990) 64–71.

[32] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofieldl, M. Shepperd, S. Webster, An Investigation of Machine Learning Based Prediction Systems, Empirical Software Engineering Research Group, 1999, http://dec.bmth.ac.uk/ESERG.

[33] J.E. Matson, B.E. Barrett, J.M. Mellichamp, Software development cost estimation using function points, IEEE Transactions of Software Engineering April (1994) 275–286.

[34] K. Maxwell, L.V. Wassenhove, S. Dutta, Performance evaluation of general and company specific models in software development effort estimation, Management Science 45 (1999) 787–903.

[35] Y. Miyazaki, K. Mori, COCOMO Evaluation and Tailoring, Eighth International Conference Software Engineering, 1985, pp. 292–299.

[36] S.N. Mohanty, Software cost estimation: present and future, Software—Practice and Experience 11 (1981) 103–121.

[38] NeuroSolutions, Users Manual, 2001, www.nd.com.

[39] R.S. Pressman, Software Engineering A Practitioner's Approach, McGraw-Hill, Englewood Cliffs, NJ, 1997.

[40] L.H. Putnam, A general empirical solution to the macro software sizing and estimating problem, IEEE Transactions on Software Engineering SE-4 (4) (1978) 345–361.

[41] L.H. Putnam, A. Fitzsimmons, Estimating software costs, Datamation 25 (10–12) (1979).

[42] L. Putnam, W. Myers, Measures for Excellence, Yourden Press Computing Series, 1992.
[45] H.A. Rubin, The Art and Science of Software Estimation: Fifth Generation Estimators, Proceedings of the Seventh Annual ISPA Conference, vol. 5, 1985.
[46] K.K. Shukla, Neuro-genetic prediction of software development effort, Information and Software Technology 42 (2000) 701–713.
[47] StatSoft Inc, Neural Networks (2001) www.StatSoft.com.
[48] P.N. SubbaNarasimha, B. Arinze, M. Anandarajan, The predictive accuracy of artificial neural networks and multiple regression in the case of skewed data: exploration of some issues, Expert Systems with Applications 19 (2000) 117–123.
[49] K. Srinivasan, D. Fisher, Machine learning approaches to estimating software development effort, IEEE Transactions on Software Engineering 21 (2) (1995) 126–137.
[50] C.R. Symons, Function point analysis: difficulties and improvements, IEEE Transactions on Software Engineering January (1988).
[52] S.S. Vicinanza, T. Mukhopadhyay, J.J. Prietula, Software-effort estimation: an exploratory study of expert performance, Information Systems Research December (1991) 243–262.
[53] C.D. Wrigley, A.S. Dexter, Software Development Estimation Models: a Review of Critique, Proceedings of the ASAC Conference, University of Toronto (1987) 125–128.