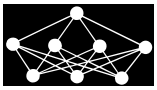

ASSESSING EMPIRICAL SOFTWARE DATA WITH MLP NEURAL NETWORKS

Petr Musilek, Jason Meltzer**

Abstract: Software measurements provide developers and software managers with information on various aspects of software systems, such as effectiveness, functionality, maintainability, or the effort and cost needed to develop a software system. Based on collected data, models capturing some aspects of software development process can be constructed. A good model should allow software professionals to not only evaluate current or completed projects but also predict future projects with an acceptable degree of accuracy.

Artificial neural networks employ a parallel distributed processing paradigm for learning of system and data behavior. Some network models, such as multilayer perceptrons, can be used to build models with universal approximation capabilities. This paper describes an application in which neural networks are used to capture the behavior of several sets of software development related data. The goal of the experiment is to gain an insight into the modeling of software data, and to evaluate the quality of available data sets and some existing conventional models.

Key words: *Neural networks, software development, effort estimation, model size, accuracy, evaluation*

Received: May 22, 2005
Revised and accepted: June 22, 2005

1. Introduction

Statistics illustrate current inability of software professionals to estimate the effort and cost of developing software products despite many existing methods of software cost modelling [22]. The results of model based estimations are still overshadowed by the estimates provided by human experts [18]. At the same time the prediction of effort to be spent on a software project is the variable most sought by project managers [10], and the variable needed throughout software lifecycle.

There have been more than three decades of active research looking for a model or function that relates effort to the size of software projects. This resulted in the development of numerous parametric models of software cost estimation such as

*Petr Musilek, Jason Meltzer
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada T6G 2V4, Tel: (780) 492–5368, Fax: (780) 492–1811, E-mail: Petr.Musilek@ualberta.ca

SLIM [19], Function Point Analysis [1], COCOMO [4], COCOMO 2.0 [5], and others. More recently, there have been attempts to construct non-parametric models using such techniques as neural networks [6], genetic programming [10], or case-based reasoning [15]. However, in general, these methods do not show a significant improvement in prediction accuracy, especially when applied across different projects and organizations.

To address the issue of accuracy and to obtain a general cost estimation model, large number of consistent data sets would have to be investigated [7]. The question is whether a large number of consistent data is available. Keeping written documentation, including project data, is a characteristics of mature software organizations [17]. However, only about one fifth of all software organizations have achieved high maturity [21], and organizations that collect data are rarely willing to share them [6].

In this paper, neural networks are used to gain insight into several available cost data sets (some of which were used to construct existing cost estimation models, e.g. COCOMO) and to capture the essence of the existing parametric models. In other words, neural networks are exploited in quite different roles than in some of the studies mentioned above, e.g. [6]. Although they are trained to provide functional approximation of available cost data, cost estimation is not the purpose for which they are built. Rather, properties of the data sets under study and characteristics of selected parametric estimation models are inferred using a set of neural network based experiments.

The paper is organized in 5 sections. Section 2 provides a brief overview of software cost estimation and neural networks. In Section 3, the experimental setup is described along with the particular data sets used in the experiments. Results of the experiments are described in full and analyzed in Section 4. Section 5 builds on the results and proceeds with a comparative study of two conventional models against a neural network-based model. Finally, Section 6 provides main conclusions and plans for future work.

2. Background

2.1 Software cost estimation

Cost and/or effort estimates for software projects are needed at many stages of the software lifecycle: preliminary estimates to determine the feasibility of a project, detailed estimates to assist with project planning, etc. The actual effort is also compared with estimated effort to reallocate resources as necessary throughout the project.

A sophisticated model of software development costs and effort takes the form of a relationship between the projected product size or required product functionality, and the development cost/effort. In addition, various productivity factors should be considered to further refine the model's prediction capabilities. Besides providing the estimation capabilities, such a model would provide a better understanding of factors affecting cost and, consequently, alleviate current software development inefficiencies by revealing directions to maximize productivity [7]. Hemstra [13] describes 29 software cost models introduced since 1966. Despite the ongoing efforts,

there is not a single model that would provide sufficient accuracy of estimates over a wide range of software projects and development organizations.

In general, software cost models predict amount of effort, E, necessary to construct a software system, as a function of the projected size, S, of the system, and a set of productivity factors, c, such as characteristics of the system to be developed, development process, personnel, etc.

$$E = f(S, c). \tag{1}$$

Traditionally, cost models have been developed using regression analysis of existing software production data. This approach requires availability of large amounts of consistent data on completed software projects including the actual values of product size and development effort, preferably complemented by a quantified description of circumstances of the development process.

Although most researchers and practitioners agree that size is the primary determinant of effort, the exact relationship between size and effort is unclear [11]. Empirical studies usually express effort as a function of size, S, with an exponential factor, b, and a multiplicative term, a,

$$E = aS^b, \tag{2}$$

however, the values of a and b vary for different models [4]. Correct structure of the model remains unclear and existing models do not achieve acceptable results when applied to data other than the data they were derived from [14].

A typical representative of parametric models constructed using regression analysis is the COnstructive COst MOdel (COCOMO) derived using data from a set of 63 projects. It is a relatively well documented and straightforward model based on inputs relating to the size of the system and a number of cost drivers that are believed to affect productivity in a log-linear fashion. The original COCOMO model was first published in 1981 [4], and then refined to its current version COCOMO 2.0 [5]. The main focus in this model is on considering the influence of cost drivers on the development effort. The values of these drivers must be first assigned/estimated along with the projected overall size of the software project or module.

The COCOMO suite is a collection of three models providing estimates with different accuracy and detail: basic, intermediate and advanced. The individual models differ in values of constants a and b. Details of this model, including the numerical values of the constants, can be found in [4], [5].

2.2 Neural Networks

Neural networks are parallel distributed information processing structures [12], typically consisting of layers of uniform processing elements called neurons. The neurons and layers are connected according to a specific architecture. Fig. 1 shows a simple architecture known as a multilayer perceptron (MLP). The number of input neurons, n, in the input layer is equal to the dimension of input vector, $\mathbf{x}$. The number of output neurons, m, in the output layer is equal to the dimension of the output vector $\mathbf{y}$. There is always at least one hidden layer with k neurons.

It has been shown [2], [9] that an MLP with one hidden layer is capable of approximating any continuous nonlinear mapping with an arbitrary precision. This makes MLP networks well-suited for a very broad class of nonlinear approximations and mappings. To approximate a mapping, an appropriate network architecture must be chosen while weights of the connections are set by a learning process. Although there are theoretical bounds for the number of hidden neurons, the optimal architecture is usually found experimentally.

It is also possible to perform this induction in the opposite direction: to interpret a network architecture in terms of properties of the mapping it approximates. The number of neurons in the hidden layer needed to realize a mapping represents the number of nonlinear terms necessary to describe the mapping. Therefore, it corresponds to the degree of nonlinearity of the relationship. Residual error after the training can also be used to evaluate the quality of the data – the lower the residual error, the higher the degree of consistency in the data.

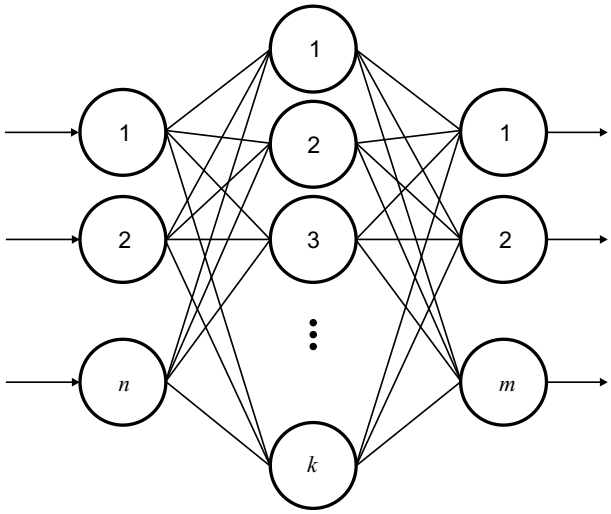

Fig. 1 *Multilayer perceptron with m-k-n architecture.*

3. Experiment Design

A series of experiments have been designed in order to gain an insight into the character of data used to build software cost models and the nature of the models themselves.

3.1 Cost data

Extensive and reliable data on software cost is very scarce in the software engineering research community. The data sets used in the experiments have been collected from other publications and their full contents can be found in the cited works [4], [8]. There have been six data sets collected in the period from late 1970s to 2001,

containing data on 161 software projects. Most of the projects are relatively small in size (80% of the projects are under 100,000 lines of code). A general description of all data sets is given in Tab. I.

Label	Name	Number of projects	S_{min} [kLOC]	S_{max} [kLOC]
1	Belady	33	4.74	712.36
2	Cocomo 2.0	15	1.90	966.00
3	Cocomo 81	63	9.10	134.47
4	NASA	18	5.00	138.30
5	US Army	15	49.80	450.00
6	Yourdon	17	7.10	132.20

Tab. I *Data sets used in experiments (S_{min} and S_{max} indicate the size of the smallest and largest project within a data set, respectively).*

3.2 The Experiments

All neural networks used in the experiments were fully connected, feed-forward, MLP networks with 2 and 3 layers corresponding to Fig. 1 with one and two hidden layers of neurons with sigmoidal activations functions. Simple backpropagation was used as the training algorithm, with parameters of 0.001 for the maximum training error (mean square error, MSE) and a learning rate of 0.3. The input and output of the networks were the size and cost to develop a software system, respectively. The size/cost data were normalized to establish a point of reference necessary to compare the results. The experiments can be divided into five groups described below.

Experiment 1 was conducted on each of the six data sets to determine the optimal number of nodes for the hidden layer of a 1-N-1 network trained to approximate each data set. Mean square error was used as the evaluation mechanism for the network configurations. Topologies containing between 2 and 65 hidden layer nodes were evaluated for each data set, which generated 6 lists of MSE values. The experiment was run five times and a simple mean of those results was used to pick the optimal topologies. A topology of 1-1-1 was found in early testing not to converge sufficiently for inclusion in this evaluation.

Experiment 2 used mean square error as the error measure. It performed leave-one-out cross-validation to determine the unbiased generalization error of each of the six data sets used to train a neural network of arbitrary configuration. The arbitrarily configuration used was a network with a 1-10-1 topology. Five iterations of the experiment were run, and the obtained values were then averaged for the final results.

Experiment 3 was a cross-validation experiment conducted with all six data sets. This experiment determined which data set displays the lowest generalization error and thus indicated which data set would be the most suitable for use in

constructing a general model. The experiment was conducted with the optimal two-layer networks found in Experiment 1 and with three-layer networks of an arbitrarily chosen topology of 1-10-10-1. The experiment was performed as follows: each data set was used individually to train a network, and the trained network was then tested with all six data sets during which MSE values were collected. This procedure was then repeated for all six data sets and for both two- and three-layer networks. The results are two 6×6 matrices of MSE values, one for two-layer and one for three-layer networks. As with the above two experiments, five runs were performed and the values averaged for the results.

Experiment 4 provided transfer function plots obtained from optimal 1-N-1 networks versus those obtained from a 1-10-10-1 network. Networks of both topologies were trained for each of the six data sets. A number of values falling between the minima and maxima of the training sets were then provided as inputs to the networks to generate the data for the transfer function plots. The resulting plots were then used to qualitatively evaluate performance of the networks in approximating the data.

Experiment 5 took into account the intermediate COCOMO model [5] including the information on development mode, type of application and all effort adjustment factors. The goal of this experiment was to find an optimal architecture of network to model the entire COCOMO data set. The experiment started with an arbitrarily chosen topology 18-10-1. It was expected that the number of hidden layers and nodes would have to be increased or decreased to achieve the optimal performance.

4. Result Analysis

The results from Experiment 1, to be used in Experiments 3 and 4, are summarized in Tab. II. A sizable variation in the number of optimal hidden nodes was expected. Correctness of this assumption can be confirmed in the table. The higher the optimal number of hidden nodes, the more nonlinearities present in the data set. And, the higher the average MSE, the more inconsitencies among the data. The data set 4 (NASA) requires a moderate number of hidden nodes to achieve the best performance in terms of an average error (an order of magnitude lower than the second lowest MSE). Thus the NASA data set is quite consistent and describes relationship between the size and effort with a moderate nonlinearity. This data set also provides a reasonable coverage of the size range as seen in Tab. I.

On the other hand, data set 3 (COCOMO 81) shows the highest MSE among all data sets. The low number of optimal hidden nodes for this data set is due to the fact that the degree of inconsistency does not allow successful training of any reasonably-sized two-layer MLP network, and the obtained "optimal" number of hidden nodes is accidental.

The outcome of Experiment 2 can be used to infer the degree of inconsistency within a data set: the higher the average error, the more inconsistent the data. The results obtained in this experiment, summarized in Tab. III, confirm that data set 4 achieved an order of magnitude smaller MSE than the second best data set. The most inconsistent data set is set 6 (Yourdon) with the average MSE two orders of magnitude higher than the best case.

Data set	k_{opt}	Avg. MSE
1	9	1.06039
2	33	0.51901
3	4	1.32412
4	22	0.07119
5	41	0.25638
6	18	0.24958

Tab. II *Optimal number of hidden nodes.*

Data set	Avg. MSE
1	0.0341766
2	0.0496576
3	0.0157605
4	0.0070931
5	0.0892731
6	0.1088250

Tab. III *Leave-one-out Cross-Validation training/testing error.*

The results of Experiment 3 can be used to evaluate the generality of the data included in a data set, i.e. the ability of a neural network trained with the data to successfully predict the effort of projects included in other data sets. The results shown in Tabs. IV and V summarize the generality of each data set used to train a two- and three-layer MLP, respectively. To allow a meaningful comparison of the results, the MSE values have been averaged across the rows and cells with a minimal average MSE set in bold type in both Tabs. IV and V.

Tab. V, summarizing the results of cross-validation performed on the three-layer networks, confirms the overall favourable properties of data set 4: neural networks trained using this data set exhibit the best generalization capabilities.

By comparing the results obtained with 2- vs. 3-layer neural networks (Tabs. IV and V respectively), one can conclude that the degree of nonlinearity inherent in the cost data requires the more complex 3-layer architecture to capture the data with higher accuracy.

	1	2	3	4	5	6	Avg.
1	0.033	0.308	0.027	0.311	0.385	0.588	0.275
2	0.033	0.737	0.027	0.159	0.645	0.564	0.361
3	0.043	0.166	0.027	0.201	0.172	0.202	0.135
4	0.043	0.519	0.040	0.099	0.538	0.571	0.302
5	0.031	0.524	0.025	0.305	0.510	0.592	0.331
6	0.048	0.146	0.033	0.157	0.121	0.180	**0.114**

Tab. IV *1-N-1 network cross-validation.*

	1	2	3	4	5	6	Avg.
1	0.001	0.125	0.016	0.140	0.094	0.123	0.083
2	0.038	0.009	0.016	0.052	0.043	0.091	0.042
3	0.049	0.251	0.011	0.101	0.205	0.168	0.131
4	0.035	0.046	0.013	0.001	0.043	0.067	**0.034**
5	0.045	0.061	0.015	0.103	0.006	0.105	0.056
6	0.066	0.115	0.037	0.053	0.133	0.016	0.070

Tab. V *1-10-10-1 network cross-validation*

Experiment 4 produced a series of graphs found in Fig. 2. The graphs illustrate the data distribution in data sets 1–6 and corresponding transfer functions obtained from 2- and 3-layer networks trained with given data sets. These graphs provide valuable information on the degree of nonlinearity of the data and on the ability of the studied network architectures to learn given data.

The results confirm the findings from previous experiments: The NASA data set (Set 4) exhibits the smallest nonlinearity and high consistency. Both 2- and 3-layer neural networks are able to approximate this data set with reasonable accuracy. Thus, the NASA data set provides an acceptable balance of small model size, with the ability to generalize and a low generalization error.

The other data sets show a high degree of nonlinearity and a substantial number of outliers. The oscillating character of transfer functions of 3-layer networks trained with data sets 5 and 6 indicate that these data are not suitable for constructing a general cost model. These difficulties might be caused by ignoring additional aspects of software products and also ignoring characteristics of the development process and team. The additional information is only available for the original COCOMO data set and therefore this data set was considered in the last experiment.

In Experiment 5, the initial architecture of the neural network was set arbitrarily to 18-10-1 (the numbers of input and output nodes are given by the dimensions of the data set). This network was able to learn to zero MSE in 100 learning cycles. Subsequently, the number of hidden nodes was decreased and the hidden layer was eventually eliminated. The resulting network has architecture 18-1 and is still able to learn the given data set with zero error. In other words, the COCOMO data set is linearly separable. This is true for a neural network trained with the entire data set. However, when the data set is divided into a training set (2/3 of the projects) and a validation set (remaining 1/3 of the projects), the situation changes dramatically. Though the training error still converges to 0, the error obtained through validation is high, as illustrated in Fig. 3.

The linear separability, achieved through adding effort adjustment factors and other characteristics, may appear as good news. However, there are two problems remaining. First, the model is clearly "over-parameterized" as it is constructed using 63 data points described by 18 parameters [4]. Second, the more parameters in the model, the more values have to be estimated. Thus the overall uncertainty of the predicted effort is the product of uncertainty inherent in all 18 parameters.

Even when these obviously practical problems were omitted, it is likely that

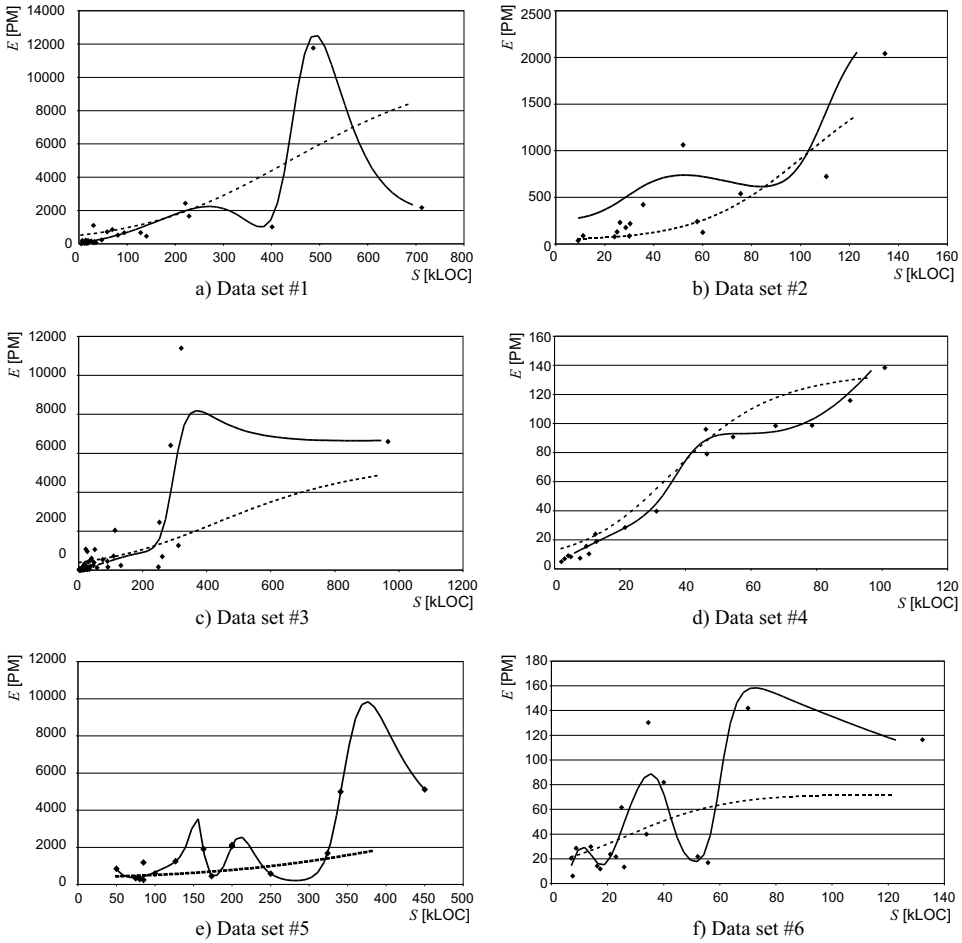

Fig. 2 *Cost data and transfer functions learned from the data by 2-layer (dashed lines) and 3-layer (solid lines) neural networks.*

inconsistency of the data (see the results of Experiment 1) together with small number of projects and large number of parameters would lead to the problem of over-fitting corrupting the generalization capabilities of any neural network.

5. Comparative study

In the previous section, the results of a series of experiments were shown and analyzed to demonstrate some important properties of several available software cost data sets. In these experiments, neural networks were used to judge the quality of the data without the goal of building any particular prediction model. It is, however, possible to use neural networks to build such models. Neural network based models can adapt to any consistent data and provide predictions outper-

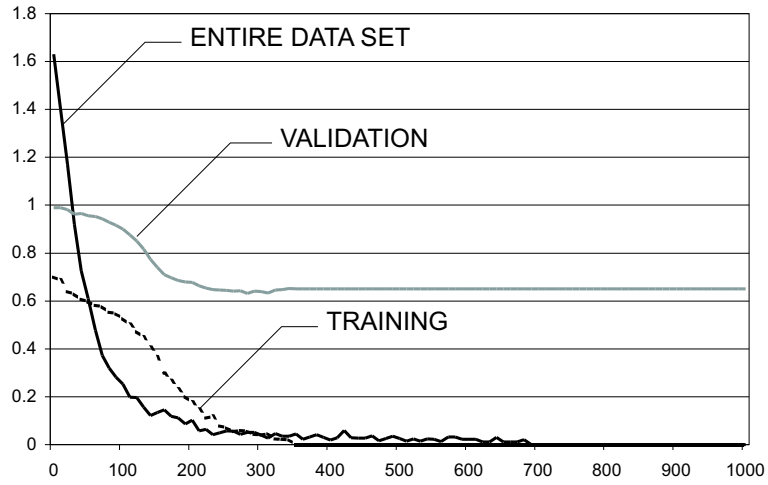

Fig. 3 *Training process of Experiment 5.*

forming traditional models [16], [10]. To demonstrate the performance of neural estimation models on data sets considered in this paper, three models were built to allow comparison.

Data from all data sets were combined to obtain a total of 161 projects. 40% of the data (65 projects) was selected at random to form a new (training) data set. This data set was then used to construct two regression models and to build a neural network prediction model.

The first regression model takes a linear form described by

$$E = a + bS, \tag{3}$$

with coefficients $a = -250$ and $b = 12.743$. The coefficient of determination for this model has value $R^2_{lr} = 0.571$.

The second model is the second order polynomial (quadratic) in form

$$E = a + bS + cS^2, \tag{4}$$

0.0398x2 - 2.488x + with coefficients $a = 304.74$, $b = -7.6773$ and $c = 0.0587$. The coefficient of determination for the quadratic model is $R^2_{qr} = 0.844$.

The third model is build using a MLP network with topology 1-22-1. The number of hidden neurons was chosen as average of optimal numbers of hidden neurons found in Experiment 1 described in Section 4.

After all three models were constructed, the entire data set (161 projects) was used to evaluate performance of each model. The results of this experiment are shown in Tab. VI.

Model	**Linear** (3)	**Quadratic** (4)	**Neural network**
pred(25) [%]	8.86	13.92	16.46
MIN error [%]	4.24	2.41	2.27
MAX error [%]	4564.78	5677.52	1410.66
AVG error [%]	510.35	582.09	111.56

Tab. VI *Estimation results for three models models.*

6. Conclusions

Estimation of effort or cost of developing software systems is an important task within any software organization. The approaches to its solution vary from expert judgement through estimation by analogy [20] or using learning models [10], to parametric cost models [4]. Each of these approaches is based (explicitly or implicitly) on knowledge of the past software projects. Therefore, the quality of any model (its ability to make correct predictions of effort) depends heavily on the quality of available data in terms of coverage and consistency.

Flexibility, objectivity, correctness, and computational economy are desirable features that make MLP networks attractive for use in data modelling applications. This paper has examined the use of such neural networks for the purpose of modelling empirical software engineering data with favorable results that can be applied to more sophisticated modelling applications.

In this study, neural models have been used to evaluate the quality of several available data sets, with quality defined as consistency within a data set and across different sets, and the number of terms (order) of the model necessary to describe the data with a reasonable precision. Evaluating the generalization capabilities of the data sets has revealed that NASA set offers the highest performance for the factors that were examined. This data set was found to have the lowest generalization error in two separate cross validation experiments and it was established that an optimal 2-layer MLP network topology for this data set does not require a large number of hidden nodes. Thus NASA data set provides an acceptable balance of a small model size with the ability to generalize and a low generalization error.

Overall, the data sets analyzed in this paper are mutually inconsistent. This indicates that cost data collected within an organization cannot be simply used to build estimation models in other organizations. In other words, building local models using local data seems to be a more valid approach than the attempts to create a universal model usable in any software organization. Neural networks are particularly suitable to build such local models as shown e.g. in [6]. Their great advantage is the ability to learn, which can be used to calibrate the model or to include new data as they become available.

The experimental results indicate that the nonlinear character of constructive cost data leads to the need for multiple hidden layers in neural networks used to model these data sets. Multiple hidden layers give the system more flexibility to model such nonlinear relations. Further validation of this hypothesis will be necessary to avoid the risk of losing generalization capabilities in exchange for higher precision of the network training.

Acknowledgement

Support provided by Natural Sciences and Engineering Research Council of Canada (NSERC) and Alberta Software Engineering Research Consortium (ASERC) is greatly appreciated. Author would also like thank Dr. Barry Boehm for providing an unlabelled sample of COCOMO 2.0 data.

References

[1] Albrecht A. J., Gaffney J. E.: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, **9**, 6, 1993, pp. 639–648.

[2] Arai N.: Mapping abilities of three-layer neural networks. In: Proc. 1989 Int. Joint Conf. Neural Networks, **1**, Washington, DC, 1989, pp. 419–423.

[3] Belady L. A., Lehman M. M.: A model of large program development, IBM Systems Journal, **15**, 1, 1976, pp. 225–252.

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

[5] Boehm B. W. et al.: Cost Models for Future Software Life Cycle Processes: COCOMO 2.0 Annals of Software Engineering Special Volume on Software Process and Product Measurement Baltzer AG Science Publishers, Amsterdam, The Netherlands, 1995.

[6] Boetticher G.: Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains, Model Based Requirements Workshop, San Diego, 2001, pp. 17–24.

[7] Briand L. C., El Emam K., Wieczorek I.: Explaining the cost of European space and military projects. In: Proc. Proceedings of the 21st international conference on Software engineering, 1999, pp. 303–312.

[8] Conte S. D., Dunsmore H. E., Shen V. Y.: Software Engineering Metrics and Models, The Benjamin Publ Menlo Park, 1986.

[9] Cybenko G.: Approximations by superpositions of a sigmoidal function, Mathemat. Control, Signals, Systems, **2**, 1989, pp. 303–314.

[10] Dolado J. J., Fernández L., Otero M. C., Ukola L.: Software Effort Estimation: The Elusive Goal in Project Management, Proceedings of the 1st International Conference on Enterprise Information Systems, Portugal, 1999, pp. 412–418.

[11] Fenton N. E., Pfleeger S. L.: Software Metrics: A Rigorous and Practical Approach, 2nd Ed., International Thomson Computer Press, 1996.

[12] Hecht-Nielsen R.: Neurocomputing, Addison-Wesley, 1991.

[13] Heemstra F.: Software Cost Estimation, Information and Software Technology, **34**, 11, 1992, pp. 627-639.

[14] Kemerer C. F.: An Empirical Validation of Software Cost Estimation Models Communications of the ACM, **30**, 5, 1987, pp. 416–429.

[15] Mair C., Kadoda G., Lefley M., Phalp K., Schofield C., Shepperd M., Webster S.: An investigation of machine learning based prediction systems, The Journal of Systems and Software, **53**, 1, 2000, pp. 23–29.

[16] MacDonell S. G., Gray A. R.: A Comparison of Modeling Techniques for Software Development Effort Prediction, In: Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems, Dunedin, New Zealand, Springer-Verlag, 1997, pp. 869–872.

[17] Paulk M. C., Curtis B., Chrissis M. B., Weber C. V.: Capability Maturity Model Version 1.1, IEEE Software, **10**, 4, 1993, pp. 18–27.

[18] Pfleeger S. L.: Making good decisions: Software Development and Maintenance Projects, Tutorial, 8th IEEE Symposium on Software Metrics, 2002.

[19] Putnam L. H.: A general empirical solution to the macro software sizing and estimating problem, IEEE Trans. on Softw. Eng., **4**, 4, 1978, pp. 345–361.

[20] Shepperd M., Schofield M.: Estimating Software Project Effort Using Analogies, IEEE Transactions on Software Engineering, **23**, 12, 1997, pp. 736-743.

[21] Software Engineering Institute, The Capability Maturity Model: Guidelines for Improving Software Process, Addison-Wesley, Reading, MA, 1995.

[22] Standish Group International, Chaos: A Recipe for Success, Standish Group Report, 1999.