A Novel Neural Network Approach For Software Cost Estimation Using Functional Link Artificial Neural Network (FLANN)

B. Tirimula Rao
Senior. Assistant professor,
Department of Computer Science and Engineering
Anil Neerukonda Institute of Technology and Sciences,
Sangivalasa, Bheemili Mdl, Visakhapatnam -531 162

B. Sameet, G. Kiran Swathi, K. Vikram Gupta, Ch. RaviTeja, S.Sumana
Department of Computer Science and Engineering
Anil Neerukonda Institute of Technology and Sciences,
Sangivalasa, Bheemili Mdl, Visakhapatnam -531 162

Summary:
Software engineering measurement and analysis specifically, cost estimation initiatives have been in the center of attention for many firms. The use of the expert judgment and machine learning techniques using neural network as well as referencing COCOMO approach to predict the cost of software have shown their strength in solving complex problems of tolerating extreme inputs but as the number of inputs increases the complexity of the neural network is maximized. A novel computationally efficient **F**unctional **L**ink **A**rtificial **N**eural **N**etwork (FLANN) is proposed for this purpose and to reduce the computational complexity so that the neural net becomes suitable for on-line applications. FLANN do not have any hidden layer; the architecture becomes simple and training does not involve full back propagation. In the course of adversity in neural networks, this dynamic neural network excellently works which will initially use COCOMO approach to predict the cost of software and uses FLANN technology with backward propagation. The proposed network processes each and every neuron crystal clear so that the entire network is completely "white box". This method gives much more accurate value when compared with others because our method involves proper training of data using back propagation algorithm which is used to train the network, becomes very simple because of absence of any hidden layer. As this method uses COCOMO as a base model, this model gives best estimate for the projects applied in Software Development Approach.

Keywords:
Functional Link Artificial Neural Networks (FLANN), Chebyshev Functional Link Artificial Neural Networks (C-FLANN), Legendre Functional Link Artificial Neural Networks (L-FLANN), Power Functional Link Artificial Neural Networks (P-FLANN),Constructive Cost Model (COCOMO), Software Cost Estimation, Effort.

1. Introduction

Software cost estimation is the prediction of hours of work and the number of workers needed to develop a project. Software cost estimation techniques fall into three general categories includes Algorithmic models, Expert Judgment and Machine Learning techniques. Among algorithmic models, COCOMO (Constructive Cost Model) is the most commonly used Algorithmic cost modeling technique because of its simplicity for estimating the effort in person-months for a project at different stages. COCOMO uses mathematical formulae to predict project cost estimation [1]. COCOMO is taken as a base model for the software cost estimation. Expert judgment is a non-structured process in which experts make decisions to the technical problems based on knowledge and experience in order to give accurate results than other techniques and doesn't require any previous historical data [2, 3, 4]. And the third technique is Machine Learning by using the neural networks. Neural networks has been found as one of the best techniques for software cost estimation [5]. Numerous researchers and scientists are constantly working on developing new software cost estimation techniques using neural networks [6].

Nasser Tadayon [6] developed an adaptive learning machine based on neural network to estimate the software cost using COCOMO model. MLP has a multi-layer architecture with one or more hidden layer(s) between its input and output layers. Node output from each layer is directly input to the successive layer nodes. The nodes in all layers (except the input layer) of the MLP contain a nonlinear function. The training of MLP starts with random initial values for its weights, and its process consists of a forward pass to propagate the input vector through the network layer by layer, and a backward pass to update the weights by the gradient descent rule. In the forward phase, the weighted sum of outputs of a lower layer is passed through the nonlinear function of a node in the upper layer to produce its output. Finally, the outputs of all the nodes of the network are computed. And then in the backward phase, the outputs of the final layer (output layer) are compared with the target values. The error obtained out of difference between them is used to update the weights of the network. After having trained for a period of time, the training error should have converged to

Manuscript received June 5, 2009
Manuscript revised June 20, 2009

a global minimum. To overcome the complexities associated with multi-layer neural network, single layer neural network can be considered as an alternative approach. But the single layer neural network being linear in nature very often fails to map the complex nonlinear problems. The classification task in data mining is highly nonlinear in nature. So solving such problems in single layer feed forward artificial neural network is almost an impossible task. To bridge the gap between the linearity in the single layer neural network and the highly complex and computation intensive multi layer neural network the Functional Link Artificial Neural Network (FLANN) is proposed by PAO [7]. The FLANN architecture uses a single layer feed forward neural network and to overcome the linear mapping, functionally expands the input vector.

FLANN architecture configured for software development effort is a two-layer feed forward network consisting of one input layer and one output layer. The FLANN generates output (effort) by expanding the initial inputs (cost drivers) and summing all to the final output layer. The activation function used here is sigmoid function. The output layer consists of one output neuron that computes the software development effort as a linear weighted sum of the outputs of the expanded inputs.

In this paper, the main focus is not only investigating the accuracy of the prediction using FLANN network but also decreasing the computational complexity of the network. The aim of this study is to verify if FLANN network can be used for prediction of effort on the basis of effort multipliers, size of the project, scale factor used in project development. To empirically evaluate the training algorithms and to find which training algorithm is suitable for the estimation purpose.

2. Cost Estimation Methods

Prediction of software development effort using Artificial Neural Networks has focused mostly on the accuracy comparison of algorithmic models rather than on the suitability of the approach for building software effort prediction systems.

The use of back propagation learning algorithms on a multilayer perceptron in order to predict development effort is well described by Witting and Finnie [8]. Karunanithi [9] reports the use of neural networks for predicting software reliability; including experiments with both feed forward and Jordon networks. Samson [10] uses an Albus multiplayer perceptron in order to predict software effort. They use Boehm's COCOMO dataset. Nasser Tadayon [6] also reports the use of a neural network with a back propagation learning algorithm. However it is not clear how the dataset was divided for training and validation purposes. Khoshgoftaar [11] presented a case study considering real time software to predict the testability of each module from source code static measures. They consider Artificial Neural Networks as promising techniques to build predictive models. Finally in the last years, a great interest on the use of Artificial Neural Networks has grown. Artificial Neural Networks have been successfully applied to several problem domains. They can be used as predictive models because they are modeling techniques capable of modeling complex functions.

3. Proposed Methodology

In this work, the Functional Link Artificial Neural Network method is used to predict software development effort (in person month) using popular algorithmic method called COCOMO.

3.1 COCOMO

The original COCOMO model was first published by Dr. Barry Boehm in 1981, and reflected the software development practices of the day. Constructive Cost Model (COCOMO) is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity.

COCOMO II is the extension to the original COCOMO. The model estimates the effort (Person-Months, PM) to develop a software system based on its projected size *SIZE*, effort multipliers EM_i and Scale Factors SF_i

$$PM = A.(SIZE)^{1.01+\sum_{i=1}^{5} SF_i}.\prod_{i=1}^{17} EM_i$$

where *PM* is the effort expressed in personmonths,

SIZE is the projected size of the software project (expressed in thousands of lines of code KLOC), *EM*i (*i* = 1,2….17) are effort multipliers, and *SFi* (*i* = 1,2….5) are exponent scale factors, A is a multiplicative constant. Scale factor is a particular characteristic of the software development that has an exponential effect of increasing or decreasing the amount of development effort, e.g. precedentedness, process maturity. The seventeen Post-Architecture effort multipliers (EM) are used to adjust the

nominal effort, Person-Months, to reflect the software product under development. These effort multipliers are grouped into four categories (product, platform, personnel and project) and their product is used to adjust the effort. A complete description of COCOMO II and its predecessors is given by Boehm [7] or in the COCOMO II model definition manual [12].

3.2 Functional Link Artificial Neural Network(FLANN)

FLANN architecture for predicting software development effort is a single-layer feed forward neural network consisting of one input layer and an output layer. The FLANN generates output (effort) by expanding the initial inputs (cost drivers) and then processing to the final output layer. Each input neuron corresponds to a component of an input vector. The output layer consists of one output neuron that computes the software development effort as a linear weighted sum of the outputs of the input Layer.

3.3 Architecture of FLANN:

The FLANN network can be used not only for functional approximation but also for decreasing the computational complexity. This method is mainly focused on functional approximation. In the aspect of learning, the FLANN network is much faster than other network. The primary reason for this is that the learning process in FLANN network has two stages and both stages can be made efficient by appropriate learning algorithms. The use of FLANN to estimate software development effort requires the determination of its architecture parameters according to the characteristics of COCOMO.

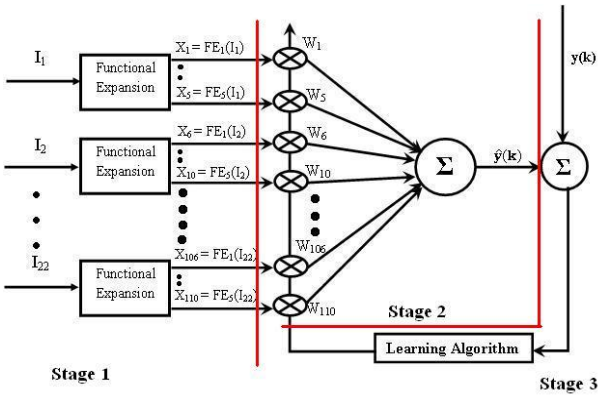

Figure 1: FLANN Architecture

A general structure of FLANN is shown in figure 1. FLANN is a single-layer nonlinear network. Let k be the number of input-output pattern pairs to be learned by the FLANN. Let the input pattern vector $\boldsymbol{X}_k$ be of dimension n, and the output y_k be a scalar. The training patterns are denoted by $\{\boldsymbol{X}_k, y_k\}$. A set of N basis functions $\boldsymbol{\emptyset}(\boldsymbol{X}_k)=[\emptyset(\boldsymbol{X}_k)\ \emptyset(\boldsymbol{X}_k)\ \bullet\bullet\bullet\ \emptyset(\boldsymbol{X}_k)\ ]^T$ are adopted to expand functionally the input signal $\boldsymbol{X}_k=[x_1(k)\ x_2(k)\ \bullet\bullet\bullet\ x_n(k)\ ]^T$. These N linearly independent functions map the n-dimensional space into an N-dimensional space, that is $\boldsymbol{R}^n \rightarrow \boldsymbol{R}^N$, $n < N$.

The linear combination of these function values can be presented in its matrix form, that is $\boldsymbol{S} = \boldsymbol{W}\boldsymbol{\emptyset}$. Here $\boldsymbol{S}_k = [S_1(k)\ S_2(k)\ \bullet\bullet\bullet\ S_m(k)]^T$, $\boldsymbol{W}$ is the $m \times N$ dimensional weight matrix. The matrix $\boldsymbol{S}_k$ is input into a set of nonlinear function $\boldsymbol{\rho(\bullet) = tanh(\bullet)}$ to generate the equalized output $\hat{Y}= [\hat{y}_1\ \hat{y}_2\ \cdots\ \hat{y}_m]^T$, $\hat{y}_j= \rho(S_j)$, j =1, 2, •••, m.

The major difference between the hardware structures of MLP and FLANN is that FLANN has only input and output layers and the hidden layers are completely replaced by the nonlinear mappings. In fact, the task performed by the hidden layers in an MLP is carried out by functional expansions in FLANN. Being similar to a MLP, the FLANN also uses BP algorithm to train the neural networks.

STAGE-1: The 22 cost factors of the validated dataset are taken as the input of the network. These factors are then expanded functionally by using the following formulas.

3.3.1 Three Types of Functional Expansion

There are three different functional expansion of the input pattern in the FLANN. They are *Chebyshev*, *Legendre* and *Power Series*, corresponding networks named as C-FLANN, L-FLANN and P-FLANN respectively.

(1) ***Chebyshev*** **polynomials are given by**:

$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1,$

$T_3(x) = 4x^3 - 3x, T_4(x) = 8x^3 - 8x^2 + 1.$

Higher order *Chebyshev* polynomials may be generated by the recursive formula given by:

$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), n \geq 2, (-1 \leq x \leq 1).$

(2) ***Legendre*** **polynomials are given by:**

$L_0(x) = 1, \quad L_1(x) = x, \quad L_2(x) = (3x^2-1)/2,$

$L_3(x) = (5x^3 - 3x)/2, \quad L_4(x) = (35x^4 - 30x+3)/8.$

Higher order *Legendre* polynomials may be generated by the recursive formula given by:

$L_n(x) = [(2n-1)x\, L_{n-1}(x)-(n-1)L_{n-2}(x)]/n, n\geq 2, (-1\leq x\leq 1).$

(3) **Power series (*x* =any value)**
$P_n(x) = x^n$, $n \geq 0$.

STAGE-2: Let each element of the input pattern before expansion be represented as z(i),1 < i < I where each element z(i) is functionally expanded as z_n (i),1 < n < N , where N = number of expanded points for each input element. In this study, N = 5 and I = total number of features in the dataset has been taken.
Expansion of each input pattern is done as follows:
$x_0(z(i)) = 1$, $x_1(z(i)) = z(i)$,
$x_2(z(i)) = 2z(i)^2 - 1$, $x_3(z(i)) = 4z(i)^3 - 3z(i)$, $x_4(z(i)) = 8z(i)^3 - 8z(i)^2 + 1$.
where, z(i), 1 < i < d , d is the set of features in the dataset.
These nonlinear outputs are multiplied by a set of random initialized weights from the range [-0.5, 0.5] and then summed to produce the estimated output y(k).All the Y(k)'s are summed to get $\hat{y}(k)$.

STAGE 3: TRAINING DATA:
This output is compared with the corresponding desired output and the resultant error for the given pattern is used to compute the change in weight in each signal path P, given by

$$\Delta W_j(k) = \mu \times xf_j(k) \times e(k)$$

where, $xf_j(k)$ is the functionally expanded input at k^{th} iteration.
If there are p patterns to be applied then average change in each weight is given by

$$\overline{\Delta W_j(k)} = \frac{1}{p}\sum_{i=1}^{p} \Delta W_j^i(k)$$

Then the equation, which is used for weight update, is given by

$$W_j(k+1) = W_j(k) + \Delta W_j(k)$$

where, $W_j(k)$ is the j^{th} weight at the kth iteration, μ is the convergence coefficient, its value lies between 0 to 1 and 1<j<J, J = M×d. M is defined as the number of functional expansion unit for one element.

$$e(k) = y(k) - \hat{y}(k)$$

where, y(k) is the target output and $\hat{y}(k)$ is the estimated output for the respective pattern and is defined as:

$$\hat{y}(k) = \sum_{j=1}^{J} xf_j(k).w_j(k)$$

where, xf_j is the functionally expanded input at kth iteration and $W_j(k)$ is the j^{th} weight at the k^{th} iteration and $W_j(0)$ is initialized with some random value from the range [-0.5, 0.5].

4. Experimental Results

The main purpose of this section is to compare the quality of effort obtained by validating standard 60 NASA projects dataset using Functional Link Artificial Neural Network against Artificial Neural Network.

4.1 Tracking Results

4.1.1 Used Data set: To validate our approach, we used the standard 60 NASA projects dataset [12]. The data is in COCOMO format so it lacks the scale factors. COCOMO measures effort in calendar months of 152 hours (and includes development and management hours).It helps software developers' reason about the cost and schedule implications of their software decisions. These cost factors are expressed in 6 stages i.e. verylow, low, nominal, high, veryhigh and extrahigh. We use 10-fold testing in which each dataset is tested one by one excluding it from the training dataset.

4.1.2 Comparative Study:The FLANN Networks are compared with the artificial neural networks interms of accuracy which is implemented using error graphs. Here we have considered Root Mean Square(RMS) Error.
Figure 2 represents the comparative study obtained by validating standard 60 NASA projects dataset using C-FLANN against ANN in terms of Root Mean Square Error and the C-FLANN minimizes error when compared to ANN for software cost estimation.

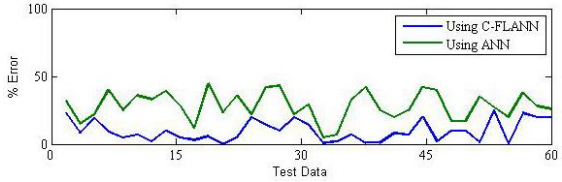

Figure 2: C-FLANN versus ANN

Figure 3 represents the comparative study obtained by validating standard 60 NASA projects dataset using L-FLANN against ANN in terms of Root Mean Square Error and the L-FLANN minimizes error when compared to ANN for software cost estimation.

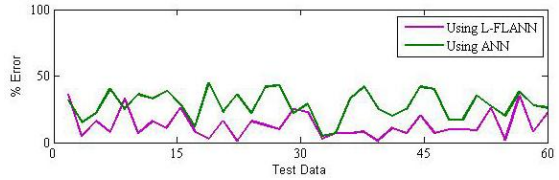

Figure 3: L-FLANN versus ANN

Figure 4 represents the comparative study obtained by validating standard 60 NASA projects dataset using P-FLANN against ANN in terms of Root Mean Square Error and the P-FLANN minimizes error when compared to ANN for software cost estimation.

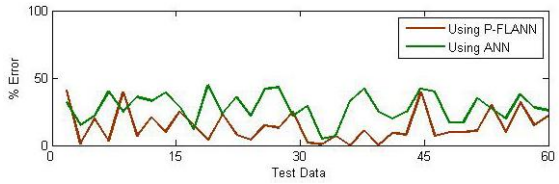

Figure 4: P-FLANN versus ANN

Figure 5 represents the comparative study obtained by validating standard 60 NASA projects dataset using C-FLANN, L-FLANN, P-FLANN against ANN in terms of Root Mean Square Error.

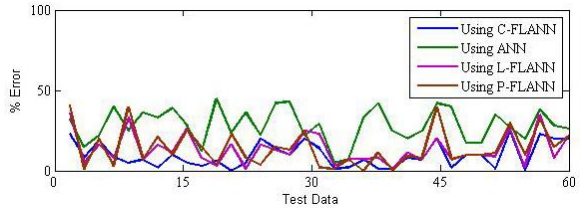

Figure 5: Error Graph

Thus, from the above graphs it is very clear that the the FLANN networks minimizes error to maximum extent than artificial neural networks and decreases complexity.

5. Conclusion

Software cost estimation is critical for software project management. Many approaches have been proposed to estimate the cost with current project by referring to the data collected from past projects. Our paper discusses an approach for the validation of the dataset for training the neural network for the software cost estimation. By combining the mathematical approach given by Nasser Tadayon [6] using COCOMO II as the base model with our validation procedure we get much accurate results when compared to the primitive ones [6]. By tracking the results with the standard ones we calculate the error percentile which is proved to be very efficient than artificial neural networks and which simplifies the operations of artificial neural networks.

6. Future Work:

Although numerous approaches have been proposed for successful project management, planning and accurate cost prediction. Cost estimators are continually faced with problems stemming from the dynamic nature of the project development process itself because software development is considered to be an intractable procedure and inevitably depends highly on several complex factors. In order to optimize the trade-offs in software effort for multiple data sets genetic algorithm [13] and fuzzy sets [14] are one of the best emerging techniques for solving real-world problems and giving accurate results and the performances with these techniques are evaluated.

References

[1] http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

[2] Jane M. Booker, Mary M. Meyer. ELICITATION AND ANALYSIS OF EXPERT JUDGMENT. Los Alamos National Laboratory.

[3] JE Zull (2002). *The Art of Changing the Brain: Enriching the Practice of Teaching by Exploring the Biology of Learning. Published:* Stylus Publishing.

[4] M. Shepperd, M. Cartwright (2001), Predicting with Sparse Data, *IEEE Trans. Soft. Eng. 27, 987-998*

[5] Karunanithi, N., etal (1992). Using neural networks in reliability prediction, *IEEE Software, 5*3-59.

[6] Nasser Tadayon. (2005). Neural Network Approach for Software Cost Estimation. *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*

[7] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley, 1989.

[8] G.Witting, and G. Finnie, "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort", J.Information Systems,1994, vol.1, no.2, pp.87-94.

[9] N. Karunanitthi, D.Whitely, and Y.K.Malaiya, "Using Neural Networks in Reliability Prediction," IEEE Software, 1992. vol.9, no.4, pp.53-59.

[10] B. Samson, D. Ellison, and P. Dugard, "Software Cost Estimation Using Albus Perceptron (CMAC)," Information and Software Technology, 1997,vol.39,pp.55-60.

[11] T.M.Khoshgoftaar, E.B.Allen, and Z.Xu, "Predicting testability of program modules using a neural network," Proc.3rd IEEE Symposium on Application-Specific Systems and Sof. Eng. Technology, 2000, pp.57-62.

[12] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases..." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. Available from http://promise.site.uottawa.ca/SERepository.

[13] Li, Y.F. Xie, M. Goh, T.N (2007). A study of genetic algorithm for project selection for analogy based software cost estimation. *IEEE International Conference on Industrial Engineering and Engineering Management*, 1256-1260.

[14] Pedrycz, W. Peters, J.F. Ramanna, S (1999). A fuzzy set approach to cost estimation of software projects. *IEEE Canadian Conference on Electrical and Computer Engineering, 2, 1068-1073.*

B. Tirimula Rao has Masters of Technology in Computer Science and Technology from Andhra University. He is currently working as a Senior. Assistant Professor in Computer Science and Engineering Department at Anil Neerukonda Institute of Technology and Sciences. His main interests lie in Image Processing, Computer Networks, Network Security, Cryptography, Neural Networks, Software Cost Estimation and Fuzzy Logic. He is a member of IEEE.

B. Sameet is a B.Tech final year student of Department of Computer Science Engineering, Anil Neerukonda Institute of Technology & Sciences. His interests include Software Cost Estimation, Image Processing and Neural Networks.

G. Kiran Swathi is a B.Tech final year student of Department of Computer Science Engineering, Anil Neerukonda Institute of Technology & Sciences. Her interests include Software Cost Estimation, Image Processing and Networks.

K. Vikram Gupta is a B.Tech final year student of Department of Computer Science Engineering, Anil Neerukonda Institute of Technology & Sciences. His interests include Software Cost Estimation and Neural Networks.

Ch. Ravi Teja is a B.Tech final year student of Department of Computer Science Engineering, Anil Neerukonda Institute of Technology & Sciences. His interests include Software Cost Estimation and Neural Networks.

S. Sumana is a B.Tech third year student of Department of Computer Science Engineering, Anil Neerukonda Institute of Technology & Sciences. Her interests include Software Cost Estimation, Wireless Communications, Genetic algorithms and Neural Networks.