Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2014, Article ID 284531, 6 pages
http://dx.doi.org/10.1155/2014/284531

Research Article

Prediction Model for Object Oriented Software Development Effort Estimation Using One Hidden Layer Feed Forward Neural Network with Genetic Algorithm

Chandra Shekhar Yadav[1] and Raghuraj Singh[2]

[1] *Department of Computer Science & Engineering, Noida Institute of Engineering & Technology, Greater Noida 201306, India*
[2] *Department of Computer Science & Engineering, Harcourt Butler Technological Institute, Kanpur 208002, India*

Correspondence should be addressed to Chandra Shekhar Yadav; csyadav@yahoo.com

Received 3 February 2014; Revised 6 May 2014; Accepted 13 May 2014; Published 3 June 2014

Academic Editor: Robert J. Walker

The budget computation for software development is affected by the prediction of software development effort and schedule. Software development effort and schedule can be predicted precisely on the basis of past software project data sets. In this paper, a model for object-oriented software development effort estimation using one hidden layer feed forward neural network (OHFNN) has been developed. The model has been further optimized with the help of genetic algorithm by taking weight vector obtained from OHFNN as initial population for the genetic algorithm. Convergence has been obtained by minimizing the sum of squared errors of each input vector and optimal weight vector has been determined to predict the software development effort. The model has been empirically validated on the PROMISE software engineering repository dataset. Performance of the model is more accurate than the well-established constructive cost model (COCOMO).

1. Introduction

The COCOMO model is the most popular model for software effort estimation. This model has been validated on large data set of projects at consulting firm, Teen Red Week (TRW) software production system (SPS) in California, USA. The structure of the model has been classified on the basis of type of projects to be handled. Types of projects are organic, semidetached, and embedded. The model structure is represented as follows:

$$\text{Effort} = a * (\text{KLOC})^{b}. \tag{1}$$

Here, a and b are domain specific parameters. For predicting the software development effort, parameters a and b have been adjusted on the past data set of various projects. Five scale factors have been used to generalize and replace the effects of the development mode in COCOMO II. There are fifteen parameters which affect the effort of software development. These parameters are analyst capability (*acap*), programmer's capability (*pcab*), application experience (*aexp*), modern programming practices (*modp*), use of software tools (*tool*), virtual memory experience (*vexp*), language experience (*lexp*), schedule constraint (*sced*), main memory constraint (*stor*), database size (*data*), time constraint for CPU (*time*), turnaround time (*turn*), machine volatility (*virt*), process complexity (*cplx*), and required software reliability (*rely*):

$$\text{Effort} = a * (\text{KLOC})^{b} * c. \tag{2}$$

KLOC is estimated directly or computed from a function point analysis and c is the product of fifteen effort multipliers:

$$\text{Effort (Months)} = a * (\text{KLOC})^{b} * (\text{EM1} * \text{EM2} * \cdots * \text{EM15}). \tag{3}$$

Proposed prediction model of software development effort estimation has been used to predict software development effort by using sixteen independent parameters such as *rely*, *data*, *cplx*, *time*, *stor*, *virt*, *turn*, *acap*, *aexp*, *pcab*, *vexp*, *lexp*, *modp*, *tool*, *sced*, and *kloc*. The past dataset has been obtained from the PROMISE site. All these sixteen parameters are used as input vector in one hidden layer feed forward neural network. Through back propagation with gradient descent training, mapping between input vectors and output vectors has been established by minimizing the sum of squared error at output layer. The optimal weight vector has been obtained through this network to predict the software development effort of another dataset of PROMISE software projects. The optimal weight vector obtained from neural network is being used as initial population in GA tool to optimize the root mean square error.

The remaining part of the paper is organized as follows. In Section 2, related works have been explained. In Section 3, mathematical model of neural network approach to effort prediction has been represented. Section 4 gives the idea of genetic algorithm in brief. Section 5 gives implementation details of prediction model. Section 6 presents result and discussion. Section 7 gives the conclusion drawn from results and future scope of the research work.

2. Related Work

Yadav and Singh obtained OHFNN 16-19-1 optimal structure for prediction of software development effort with best root mean square as 0.00149074 at the learning rate 1.01 and momentum 0.7 in one million epochs [1]. Yadav and Singh modified COCOMO II by introducing some more parameters for predicting the software development effort [2]. Kumar et al. considered mean of square distributed error as fitness function for measuring the performance of multilayer feed forward neural network in terms of accuracy, and epochs [3]. Kumar et al. proposed a model using particle swarm optimization (PSO) for tuning the parameters of basic COCOMO model to predict the software development effort accurately considering only KLOC parameter [4]. Praynlin and Latha confirmed that back propagation algorithm is more efficient than a recurrent type neural network [5]. Kumar et al. used real coded genetic algorithms and fuzzy lambda tau methodology for reliability analysis of waste clean-up manipulator [6]. Shrivastava and Singh evaluated performance of feed forward neural network with the help of three algorithms such as back propagation, evolutionary algorithm, and hybrid evolutionary algorithm for hand written English alphabets [7]. Sheta and Al-Afeef developed genetic programming model utilizing line of code and methodology to predict the software development effort precisely compared to other models [8]. Sheta proposed modified version of COCOMO model using genetic algorithms (GAs) to explore the effect of the software development adopted methodology in effort computation [9].

Reddy and Raju used single layer feed forward neural network with back propagation learning algorithm and resilient back propagation algorithm for predicting the software development effort accurately [10]. Reddy and Raju proposed multilayer feed forward neural network with back propagation learning algorithm by iteratively processing a set of training samples and comparing the network's prediction with the actual effort [11]. Singh and Dhaka analyzed the performance of back propagation algorithm with changing training patterns and the momentum term in the feed forward neural networks [12]. Tian and Noore used genetic algorithm to globally optimize the number of the delayed input neurons and the number of neurons in the hidden layer of the neural network architecture [13]. Jun and Lee used quasioptimal case-selective neural network for software development effort estimation and adopted the beam searched technique and devised the case-set selection algorithm to find the quasioptimal model from the hierarchy of reduced neural network models [14]. Burgess and Lefley evaluated the potential of genetic programming (GP) in software effort estimation when compared with existing approaches, in terms of accuracy and ease of use [15]. Shukla presented a new genetically trained neural network model for predicting the software development effort [16]. Khoshgoftaar et al. used neural network as tool for predicting the number of software development faults [17]. Here principal components are linear combinations of sixteen independent parameters X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, and X_{16}:

$$\begin{aligned} P_1 &= w_{11} \times X_1 + w_{12} \times X_2 + \cdots + w_{1\,16} \times X_{16}, \\ P_2 &= w_{21} \times X_1 + w_{22} \times X_2 + \cdots + w_{2\,16} \times X_{16}, \\ &\vdots \\ P_k &= w_{k1} \times X_1 + w_{k2} \times X_2 + \cdots + w_{k\,16} \times X_{16}. \end{aligned} \tag{4}$$

In principal component analysis, preprocessing of the dataset has been done. These sixteen attributes of project such as *rely*, *data*, *cplx*, *time*, *stor*, *virt*, *turn*, *acap*, *aexp*, *pcab*, *vexp*, *lexp*, *modp*, *tool*, *sced*, and *kloc* are correlated with the development effort. For minimizing the difference between desired output and actual output, weights have been adjusted repetitively by using ANN. The sum of squared errors on the training data set has been minimized by finding a vector of connection weights that is called network learning. The different network architecture has been trained by the standard error back propagation algorithm at different learning rate and at different momentum, having minimum sum of squared errors. Having minimum sum of squared errors is a training stopping criterion.

Multilayer networks are more powerful than single-layer networks for adjusting the weights. First appropriate transfer function is chosen by the designer after that parameters weight w and bias value b will be adjusted by some learning rule for minimizing the difference between desired output and actual output. Log-sigmoid function takes the input between plus and minus infinity and output varies into the range 0 to 1. Expression of log-sigmoid function is given as follows:

$$a = \frac{1}{1 + e^{-n}}. \tag{5}$$

The log-sigmoid transfer function is commonly used in multilayer networks which are trained using back propagation algorithm due to its differentiable nature.

3. Effort Prediction Model Using One Hidden Layer Feed Forward Neural Network (OHFNN)

OHFNN with gradient descent back propagation learning method has been used in this model for estimating the object oriented software development effort. Let us consider input vector $X_k^T = (x_1; x_2; \dots x_n)$ where $n = 16$ and output vector $D_k^T = (d_1; d_2; \dots d_p)$. The neural network can be trained by using the input and output vector mapping. P is set of Q training vector pairs:

$$P = \{X_k, D_k\}_{k=1}^{Q}, \tag{6}$$

$X_k \in R^n$, $D_k \in R^p$, where $n = 16$, $p = 1$, and $Q = 40$.

Here net_k generates an output signal vector $f(Y_k)$ and Y_k is vector of activations of output layer neuron.

Error at kth training pair (X_k, D_k) is as follows:

$$E_k = D_k - f(Y_k), \tag{7}$$

where

$$E_k = \left(e_1^k, \dots e_p^k\right)^T = \left(d_1^k - f\left(y_1^k\right), \dots, d_p^k - f\left(y_p^k\right)\right)^T. \tag{8}$$

Squared error is sum of squares of each individual output error e_j^k; that is,

$$\xi_k = \frac{1}{2}\sum_{j=1}^{p}\left(d_j^k - f\left(y_j^k\right)\right)^2 = \frac{1}{2}E_k^T E_k. \tag{9}$$

The mean square error (mse) is computed over the entire training set P:

$$\text{mse} = \frac{1}{Q}\sum_{k=1}^{Q}\xi_k. \tag{10}$$

The weights between hidden and output layer are updated as

$$w_{hj}^{k+1} = w_{hj}^k + \Delta w_{hj}^k. \tag{11}$$

and the weights between input and hidden layer are updated as

$$w_{ih}^{k+1} = w_{ih}^k + \Delta w_{ih}^k, \tag{12}$$

where Δw_{hj}^k and Δw_{ih}^k are weight changes computed in previous step.

Now the weights have been updated in output and hidden layers by the following equations:

$$\begin{aligned} \Delta w_{hj}^{k+1} &= \Delta w_{hj}^k + \eta\delta_X^k f\left(z_h^k\right), \\ \Delta w_{ih}^{k+1} &= \Delta w_{ih}^k + \eta\delta_X^k x_i^k. \end{aligned} \tag{13}$$

We can introduce the momentum into back propagation with the help of the following equations:

$$\Delta w_{hj}^k = \eta\delta_j^k f\left(z_h^k\right) + \alpha\Delta w_{hj}^{k-1}, \qquad \Delta w_{ih}^k = \eta\delta_h^k x_i^k + \alpha\Delta w_{ih}^{k-1}. \tag{14}$$

Back propagation propagates changes back because it can do substantial good thing. The change in O_j should be proportional to $O_k(1-O_k)$ the slope of the threshold function, at node k. The change to O_j should also be proportional to W_{jk} the weight on the link connecting node j to node k. Summing over all nodes in layer k, $\beta_k = \sum_k w_{jk}O_k(1-O_k)\beta_k$. At the output layer, the benefit has been given by the error at the output node. The output layer z will be benefited as $\beta_z = d_z - o_z$. Here a rate parameter r has been introduced for controlling the learning rate. So change in w_{ij} is proportional to r; that is, $\Delta w_{ij} = rO_iO_j(1-O_j)\beta_j$ and $\beta_j = \sum_k W_{jk}O_k(1-O_k)\beta_k$ for nodes in hidden layers and $\beta_z = d_z - o_z$ for nodes in the output layer. The output of the network is compared with desired output; if it deviates from desired output, the difference between actual output and the desired output is propagated back from the output layer to previous layer to modify the strength or weight of connection.

4. Genetic Algorithm (GA)

GA is a type of evolutionary concept generally used to solve optimization problems. GA is called a global optimizer. GA is based on the principles of evolution and inheritance. GA system maintains a population of potential solutions. It has some selection process based on fitness of individuals and a set of biologically inspired operators. GA consists of both a local search operator such as crossover and a global search operator such as mutation. In an evolutionary theory, only the fit individuals in a population are likely to survive and generate offspring, and their biological traits have been inherited in the next generations. In the large search space GA is much better than the conventional search and optimization techniques due to its parallelism and random search implemented by recombination operators. The following three steps are followed for GA to solve any given problem [18].

(1) Create an initial population of potential solutions to the given problem randomly.

(2) Repeatedly perform the following subsets for each generation until a termination condition has been satisfied.

   (i) Compute the fitness value of each individual in the population and save the best individual of all previous populations.

   (ii) Create the new population by applying the genetic operators such as selection, crossover, and mutation.

   (iii) Replace the current population with the new population.

(3) Individual with best fitness value is the optimum solution.

A brief description of various operators used in GA and some of the basic terminologies is given.

Selection. This operator selects fit individuals from the population for reproduction to generate offspring in the next generation. Selection is based on fitness value.

Crossover. This operator generates offspring from each pair of individuals. Each individual contributes a portion of its genetic information to each offspring.

Mutation. This operator randomly changes a tiny amount of genetic information in each offspring.

Chromosome. The complete genetic description of an individual is described as chromosome. It is a collection of basic features called genes.

Gene. This is a single feature within a chromosome. Gene may take any of several values called alleles.

Allele. Allele is a particular value that may be taken by a gene.

Population. A number of chromosomes form a single population.

Objective Function. This is a function that is considered for minimization or maximization of certain criterion.

Fitness. This is a measure of how well a parameter set performs.

Schema. This is a collection of genes in a chromosome having certain specified values.

Functioning of GA can be visualized as a balanced combination of exploration of new regions in the search space and exploitation of already sampled regions. By choosing the right control parameters such as the crossover and mutation probabilities and population size, performance of GA can be measured. The chromosome level representation is called the genotype. All the information that is necessary to construct an organism has been resided in genotype. The organism is called phenotype [16].

5. Implementation Details of OHFNN Prediction Model Using GA

OHFNN with 16 input neurons, 19 hidden layer neurons to develop input output mapping, and 1 output neuron to predict development effort in person-months has been taken. GA has been used to solve the problem of optimizing the weights of OHFNN 16-19-1 in order to minimize the mean squared error over a training PROMISE data set [19]. Here OHFNN 16-19-1 is called the phenotype, and the string of weights of OHFNN 16-19-1 is called the genotype. Genotype is a data structure which represents information about the phenotype and which is encoded for use in GA. Since 16 neurons and one bias value are at input layer, 19 neurons and one bias value are at hidden layer, and 1 neuron is at output layer, so weight vector from input to hidden layer is 17×19, and weight vector from hidden to output layer is 20×1. OHFNN 16-19-1 consists of 343 weights connecting various layers. These 343 weights are encoded in a chromosomal string. In this optimization problem fitness function is represented in terms of root mean square error (rmse) shown as follows:

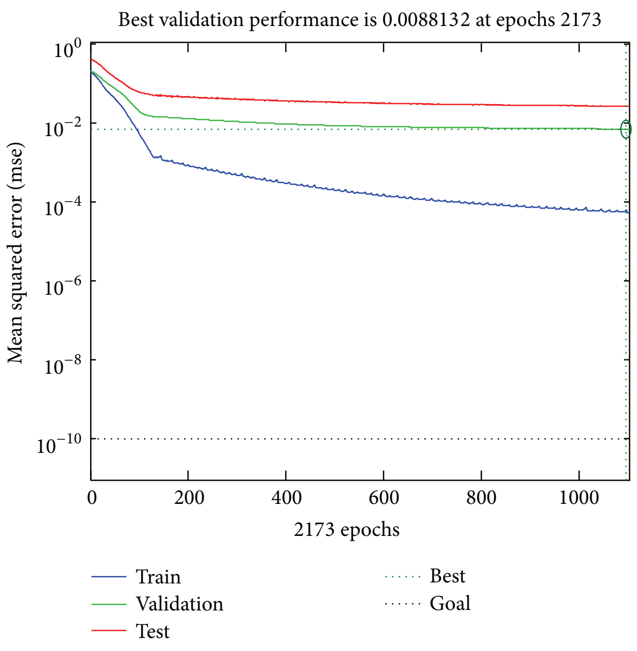

FIGURE 1: Performance graph of OHFNN 16-19-1 with traingda.

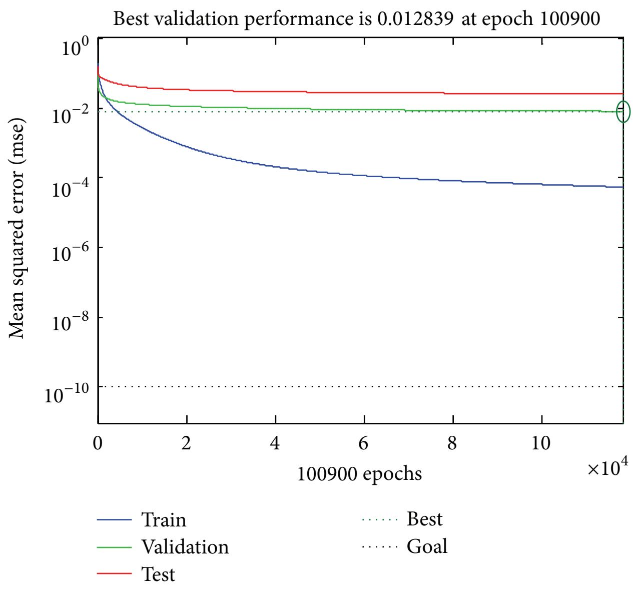

FIGURE 2: Performance graph of OHFNN 16-19-1 with traingdm.

$$\text{rmse} = \text{sqrt}\left(\frac{1}{Q}\sum_{k=1}^{Q}\xi_k\right). \qquad (15)$$

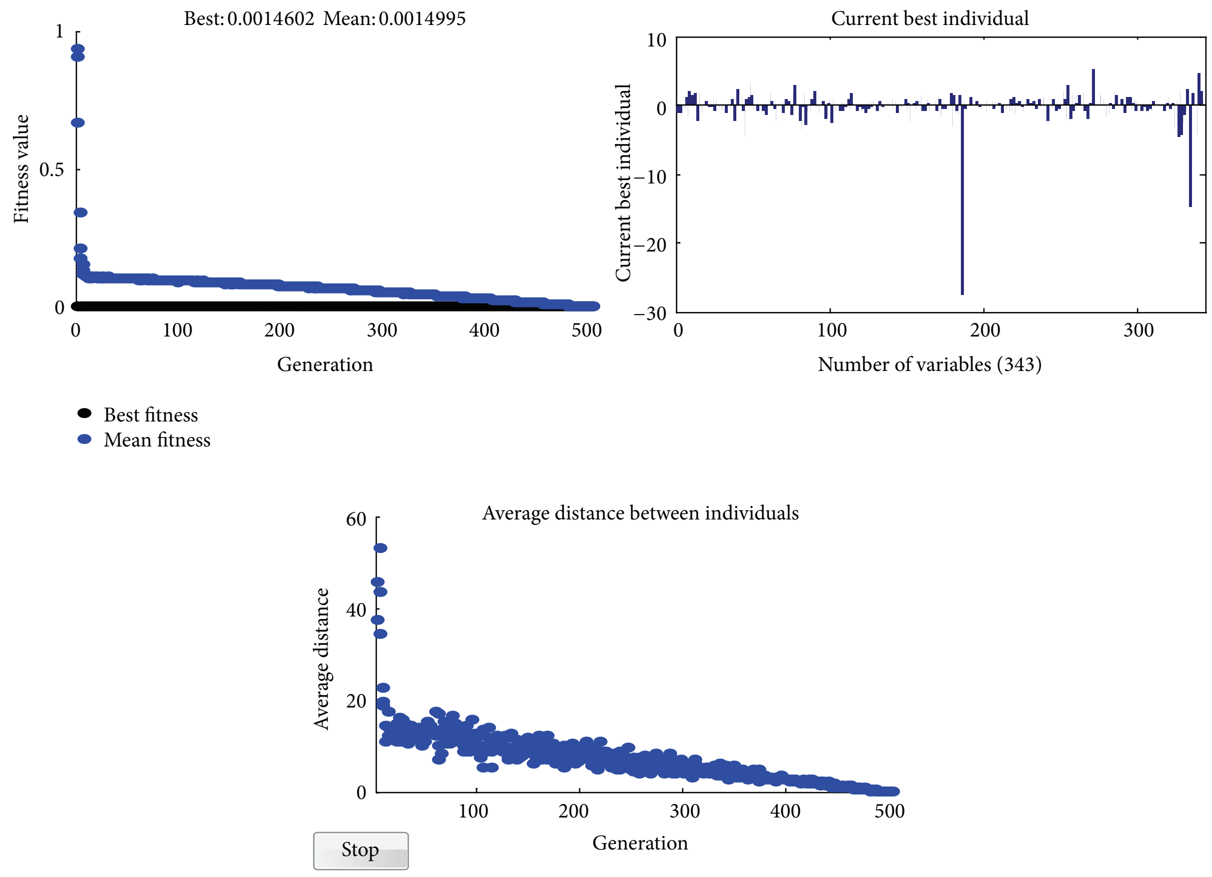

FIGURE 3: Optimized value of root mean square error for OHFNN 19-16-1 using GA.

TABLE 1: Control parameters of GA for adjusting the weight of connections in OHFNN 16-19-1 for optimizing the root mean square error.

Parameters	Value
Population size	10000
Elite count	4
Crossover fraction	0.7000
Generations	500
Initial population	[10000 × 343 double]
Selection function	Roulette
Crossover function	Heuristic
Number of variables	343
Scaling function	Rank

The real valued coding scheme has been used to form a string. This predictor has been developed on Intel core 2 Duo CPU 2.10 GHz, 2GB RAM, Windows 7 32-bit OS using NNtool and GA Tool of MATLAB. In this predictor, first we obtain best four weight vectors after training OHFNN 16-19-1 on PROMISE data set. These four weight vectors are solutions in initial population of GA. Excellent results can be obtained by using the control parameters of GA given in Table 1.

6. Results and Discussion

By varying the number of neurons at hidden layer of OHFNN architecture, the optimal neural architecture of OHFNN is 16-19-1 for traingdm and traingda training methods of NNtool. The performance graphs of OHFNN 16-19-1 with traingda and OHFNN 16-19-1 with traingdm are shown in Figures 1 and 2, respectively. Best validation performance of OHFNN 16-19-1 with traingda is 0.0088132 at epoch 2173 and the best validation performance of OHFNN 16-19-1 with traingdm is 0.012839 at epoch 1,00,900. During the analysis of this work it has been found that development effort of some projects is not predicted precisely. Research work has been carried out to change the proposed algorithm for better results in all cases. In [1] best root mean square error is 0.00149074 for network architecture OHFNN 16-19-1 at learning rate 1.01 and momentum 0.7. Gradient descent never guarantees that root mean square error obtained is a global one. For exploring the problem in global search space, GA has been used to optimize the fitness function. This fitness function is written in terms of root mean square error. Weight vector obtained after training the neural network has been used as input for the fitness function. Using the operators such as selection, crossover, and mutation operator of GA, root mean square error can be further optimized. Root mean square error is 0.0014602 after 500 generations using 10000 populations as shown in Figure 3. Control parameters of GA for the above are represented in Table 1.

7. Conclusion and Future Scope

In this research work, by a large number of simulation works, OHFNN 16-19-1 architecture has been obtained to predict the development effort accurately using GA. Performance index of OHFNN-GA prediction model depends not only on the architecture of network and learning algorithm for training but also on the crossover and mutation probabilities and population sizes. In this study, OHFNN 19-16-1 has been fixed with both training algorithms for having common platforms in the comparison of the performance. In the future the other neural network like radial basis function (RBF) with GA would be used for the prediction model. Binary associative architecture (BAM) with GA can also be used for a better result. Two hidden layer feed forward neural networks (THFNN) with GA can be used for further optimizing the rmse. Particle swarm optimization (PSO) can be combined with neural network architecture to predict the object oriented software development effort precisely. Other attributes of the object oriented software can also be predicted using this model. This nonparametric model can be used for establishing relationship between input vector and output vector with the help of weight vector.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] C. S. Yadav and R. Singh, "Implementation of prediction model for object oriented software development effort estimation using one hidden layer neural network," *International Journal of Advanced Computer Research*, vol. 4, no. 14, part 1, pp. 157–166, 2014.

[2] C. S. Yadav and R. Singh, "Tuning of COCOMO II model parameters for estimating software development effort using GA for PROMISE project data set," *International Journal of Computer Applications*, vol. 90, no. 1, pp. 37–43, 2014.

[3] S. Kumar, M. P. Singh, R. Goel, and R. Lavania, "Hybrid evolutionary techniques in feed forward neural network with distributed error for classification of handwritten Hindi 'SWARS'," *Connection Science*, vol. 25, no. 4, pp. 197–215, 2013.

[4] A. Kumar, C. S. Yadav, and P. Singh, "Parameter tuning of COCOMO modelfor software effort estimation using PSO," in *Proceedings of the 1st International Conference on Innovations and Advancements in Information and Communication Technology (ICIAICT '12)*, pp. 99–105, 2012.

[5] E. Praynlin and P. Latha, "Performance analysis of software effort estimation models using neural networks," *International Journal of Information Technology and Computer Science*, vol. 5, no. 9, pp. 101–107, 2013.

[6] N. Kumar, J. Borm, and A. Kumar, "Reliability analysis of waste clean-up manipulator using genetic algorithms and fuzzy methodology," *Computers and Operations Research*, vol. 39, no. 2, pp. 310–319, 2012.

[7] S. Shrivastava and M. P. Singh, "Performance evaluation of feed-forward neural network with soft computing techniques for hand written English alphabets," *Applied Soft Computing Journal*, vol. 11, no. 1, pp. 1156–1182, 2011.

[8] A. F. Sheta and A. Al-Afeef, "A GP effort estimation model utilizing line of code and methodology for NASA software projects," in *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA '10)*, pp. 290–295, IEEE, Cairo, Egypt, December 2010.

[9] A. F. Sheta, "Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects," *Journal of Computer Science*, vol. 2, no. 2, pp. 118–123, 2006.

[10] C. S. Reddy and K. V. S. V. N. Raju, "An optimal neural network model for software effort estimation," *International Journal of Software Engineering*, vol. 3, no. 1, pp. 63–78, 2010.

[11] C. S. Reddy and K. V. S. V. N. Raju, "A concise neural network model for estimating software effort," *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, pp. 188–193, 2009.

[12] M. P. Singh and V. S. Dhaka, "Handwritten character recognition using modified gradient descent technique of neural networks and representation of conjugate descent for training patterns," *International Journal of Engineering, Transactions A: Basics*, vol. 22, no. 2, pp. 145–158, 2009.

[13] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering and System Safety*, vol. 87, no. 1, pp. 45–51, 2005.

[14] E. S. Jun and J. K. Lee, "Quasi-optimal case-selective neural network model for software effort estimation," *Expert Systems with Applications*, vol. 21, no. 1, pp. 1–14, 2001.

[15] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? A comparative evaluation," *Information and Software Technology*, vol. 43, no. 14, pp. 863–873, 2001.

[16] K. K. Shukla, "Neuro-genetic prediction of software development effort," *Information and Software Technology*, vol. 42, no. 10, pp. 701–713, 2000.

[17] T. M. Khoshgoftaar, A. S. Pandya, and H. B. More, "A neural network approach for predicting software development faults," in *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, pp. 83–89, IEEE, October 1992.

[18] S. Kumar, *Neural Networks: A Classroom Approach*, Tata McGraw Hill Education Private, New Delhi, India, 2nd edition, 2004.

[19] http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff.