International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 17 (2015) pp 37813-37823

An Improved Neural Network Learning Algorithm Using Glow-worm Swarm Optimization for Software Defect Prediction

V.Jayaraj
Bharathidasan University, Trichy, Tamil Nadu,

N. Saravana Raman,
Research Scholar, Bharathidasan University Trichy, Tamil Nadu
nsaravanaraman@gmail.com

Abstract
Defects defined in disparate terms are aberrations from specifications or ardent expectations leading to procedure failures. Classification and prediction extracting models to describe defect data classes or predict future defect trends is the foundation for Software Defect Prediction. Classification algorithms that identify software defects or faults based on software metrics have a big role in software risk management. Popular software metrics including Line of Code, Cyclomatic complexity and its derivatives are easy to measure and can be automated. In this paper, a cubic spline based improved activation function for Multi-Layer Perceptron Neural Networks to classify defects based on software metrics is proposed. A Glow-worm Swarm Optimization algorithm to train MLPNN is also proposed to reduce the mean squared error. The proposed technique is evaluated with KC1 dataset with and shows improved classification accuracy compared to Multi-Layer Perceptron Neural Network.

Keywords: Software Defect Prediction (SDP), Software metrics, Multi-Layer Perceptron Neural Network (MLPNN), Back-propagation algorithm, Glow Swarm Optimization.

Introduction

Studies show that most defects are found in only few software modules. Such modules cause software failures, increase development and maintenance cost, and decrease customer satisfaction [1]. So, prediction of defect-prone software modules helps software developers focus on quality assurance and allocate effort and resources efficiently. This leads to substantial improvement in software quality [2]. Identification of defect-prone software modules is achieved through binary prediction models that classify a module as either defective or not. Such prediction models use static product metrics associated with defects as independent variables [3].

To predict software modules defect-proneness, software metrics provide a quantitative description of program attributes. Most software metrics are developed for this, and most are based on size and complexity. Lines Of Code (LOC) are a common size metric for defect prediction while McCabe and Halstead are mostly complexity metrics. Many works were undertaken to locate correlation between software metrics and defect proneness by constructing different predictive models including statistical methods, discriminant analysis, parametric models, machine learning methods, and mixed algorithms [4].

As relationship between software metrics and defect-proneness in software modules is complicated and nonlinear, machine learning methods like Neural Networks (NN) are adequate for the problem compared to traditional linear models [5]. NN approaches were a universal adaptor for any non-linear continuous function with arbitrary accuracy [6]. Many papers in literature state that NN offer promising approaches for software reliability estimation and modelling [7-12]. NN, are a collection of fast processing and computing nodes called artificial neurons designed based on the study and behaviour of biological neurons. They are connected in a specific manner in a layer like structure called NN architecture. It is an output based computing technique. It is a technology used for optimizing problems. NNs consist of many computational unit layers, interconnected in a feed-forward way. NN is applied to estimate parameters of a formal model and learn the process to predict future outcomes. It revealed that a feed forward network is useful for prediction. Back-error propagation is a widely used NN paradigm applied in application studies in varied areas [13].

An emphasis in NN research is on learning algorithms and architectures. Finding a suitable network structure and optimal weight values, make NN's design difficult optimization problems. Multi-Layer Perceptron Neural Networks (MLPNN) model has 3 layers. The first is an input layer, the next is a hidden layer, and last is an output layer. Such artificial networks have many properties making them suit complex pattern classification problems. But, the success of their application to real world problems depends on a training algorithm. They need this to locate a nearly globally optimal set of weights in a short time. Traditional MLPNN training algorithm called back propagation, often finds a good set of weights in reasonable time [14]. Backpropagation is a variation of gradient search and the key to backpropagation is a method to calculate error gradient regarding weights for a given input by propagating an error backwards through a network. But, backpropagation has drawbacks like getting stuck in local minima and computational complexity [14].

During learning, weights and other parameters are updated after every input/output pair representation requiring a modification of weight for each layer thereby expanding search space for finding optimized weights and bias matrix. Many global optimization methods are proposed to train MLPNN to overcome gradient based algorithms disadvantages. Nature inspired meta-heuristics like Genetic Algorithm (GA), Particle Swarm Optimization (PSO)

algorithm, Artificial Bee Colony (ABC) algorithm are applied successfully for training MLPNN [15-17].

Software criticality in present day applications has led to an increase in work being carried out in SDP. Use of intelligent NN and hybrid techniques in place of traditional statistical techniques has shown great improvement in software reliability prediction recently. This study investigates MLPNN classification accuracy for an SDP and proposes an improved cubic spline based activation function for MLPNN. This study uses Glow-worm Swarm Optimization (GSO) algorithm to train MLPNN to classify defects

Related Work

An artificial NN based approach for software reliability estimation and modelling was proposed by Su and Huang [7]. The NN approach builds a dynamic weighted combinational model. The proposed model's applicability is proved through real software failure data sets. Results from experiments show that the new model has fairly accurate prediction capability.

Three cost-sensitive boosting algorithms to boost NN for SDP were proposed by Zheng [8]. The first algorithm based on threshold-moving attempts to shift a classification threshold to not-fault-prone modules so that more fault-prone modules are classified correctly. The other two weight-updating based algorithms incorporate misclassification costs into a weight-update rule in boosting procedure so that the algorithms boost more weights on samples associated with misclassified defect-prone modules. The 3 algorithms performances are evaluated by 4 datasets from NASA projects regarding a singular measure and normalized expected cost of misclassification. Results suggest that threshold-moving is best choice to build cost-sensitive SDP models with boosted NN among the 3 algorithms studied, specially for datasets from projects developed by object-oriented language.

Hu et al., [9] proposed a robust recurrent NN modelling for software fault detection and correction prediction in which the author suggested the following approach. First, recurrent NN was applied to model the processes. A systematic network configuration approach was developed in the framework, with GA according to prediction performance. An extra factor characterizing dispersion of prediction repetitions was incorporated into performance function to ensure robust predictions. Comparison with feed forward and NN analytical models was developed regarding a real data set.

An improved NN dynamic prediction model for software reliability evaluation was proposed by Ma et al., [10]. The new model has a selective model framework, using selected models to build a combination model. To evaluate the proposed model's performance experiments on real failure data was conducted. Results showed that the new model improved prediction capability due to well-selected data models. It was seen that the improved structure adopted well selected data models from data models to build a combination model.

A software reliability prediction model using feed-forward NN for better reliability prediction through back-propagation algorithm was described by Singh and Kumar [11] who discuss issues of network architecture and data representation methods. Comparative analysis between the new approach and 3 popular software reliability growth prediction models using 7 different failure datasets from standard software projects testing the proposed method's validity was presented. A numerical example was cited to illustrate results that revealed major improvement using artificial neural network over conventional statistical models based on non-homogeneous Poisson process.

Bayesian networks to model the relationships among metrics and defect proneness on multiple data sets were proposed by Okutan and Yildiz [12]. Bayesian networks determine probabilistic influential relationships among software metrics and detect defect proneness. Also, in addition to metrics used in Promise data repository, two more metrics, i.e. NOD for the number of developers and LOCQ for the source code quality are defined.

The advantages and disadvantages in the existing system obtained from the literature survey are tabulated.

Author	**Technique proposed**	**Disadvantage**	**Improvement direction**
Su and Huang	Neural Network	Parameter optimization not done	Heuristic methods can be used to solve the NP problem
Zheng	Neural Network	Thresh holding value is ambiguous	Optimization techniques to be investigated
Hu et al.,	Recurrent Neural Network with GA	Training optimization not done	Move away from traditional training technique
Singh and Kumar	Artificial Neural Network	Number of hidden layer is large	Reduce and optimize

From literature survey it can be seen that Neural Networks perform better than Bayesian network based classifiers. However solutions in Neural Networks can be suboptimal due to the non-deterministic polynomial characteristics of its parameters. Most work concentrated on the Neural Network with back propagation algorithm which is shown to give sub optimal solution. In this work it is proposed to optimize the training parameters of

In this work we propose a new activation function based on the spline algorithm and improve the training algorithm by proposing a glow swarm optimization technique. This work assumes significance as even a small improvement in classification accuracy has a direct impact in cost savings for the software industry.

Materials and Methods

In this work a novel MLPNN activation function based on the principles of cubic spline is proposed to replace the popular sigmoidal or tanh activation function. Since back propagation algorithms have high training error a Glow-worm Swarm Optimization (GSO) algorithm to train MLPNN is proposed. The improved classifier is tested on the KC1 dataset and its performance measured.

KC1 dataset, a public, NASA Metrics Data Program [18] verifies and improves predictive software engineering models. KC1 is a C++ system implementing storage management for ground data receipt and processing. Dataset includes McCabe and Halstead features code extractors. Measures are module based. KC1 dataset has 2109 instances, 22 different attributes including 5 different LOC, 12 Halstead metrics, 3 McCabe metrics, a branch count and 1 goal-field. Dataset attribute information is McCabe's line count of code, design complexity, effort, program length, cyclomatic complexity, Halstead, total operands, class, and others.

Back propagation is a popular supervised learning algorithm with multi-layered feed-forward networks [19, 20]. Inputs are fed to input layer and propagated through layers to get an output. The output signal is computed with weights, bias, and an activation function. Propagation rule trains a network by back propagating errors and changing nodes weights. The difference between output obtained and desired output is the error.

$if\ x_i = input$ of neuron i at input layer

$t_i =$ target output vector $(t_1,\ldots,t_k,\ldots,t_m)$

$\alpha =$ learning rate

$w_{oj} = bias$ on j^{th} hidden unit

$w_{ij} =$ weight on i^{th} neuron at hidden layer j

$n_{ij} =$ neuron i in j^{th} hidden layer

The input to the first hidden layer n_{i1} is given by

$$n(in)_{i1} = w_{01} + \sum_i x_i w_{i1}$$

The output of neuron n_{i1} is given by

$$n_{i1} = f(n(in)_{i1})$$

The process is extended to all hidden units. The net input to y_k to output layer is computed by

$$y(in)_k = w_{0k} + \sum_i x_i w_{ik}$$

The output is given by

$$y_k = f(y(in)_k)$$

Each output unit $y_k (k = 1\ to\ m)$ whose target is, error correction is given by

$$\delta_k = (t_k - y_k) f'(y(in)_k)$$

Based on the error obtained, weights and bias are updated such that

$$\Delta w_{ik} = \alpha \delta_k n_j$$

$$\Delta w_{0k} = \alpha \delta_k$$

is sent to all the hidden layers

Each hidden unit n_{ij} sums its delta inputs from output units so that

$$\delta(in)_j = \sum_{i=1}^{m} \delta_j \omega_{ij}$$

The term $\delta(in)_j$ is multiplied with derivative of $f(n(in)_j)$ to calculate error term:

$$\delta_j = \delta(in)_j f'(n(in)_j)$$

Bias and weights are propagated and updated in every hidden layer. Bias and weights of each output unit is updated as follows:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

$$w_{0k}(new) = w_{0k}(old) + \Delta w_{0k}$$

Each hidden unit updates its bias and weights:

$$w_{ij}(new) = w_{ij}(old) + wv_{ij}$$

$$w_{0j}(new) = w_{0j}(old) + wv_{0j}$$

The above is continued for a specified number of epochs or when actual output equals target output. The learning rate,α, affects BPN convergence. A larger value of α may speed up convergence but result in overshooting. A smaller value ofα has the reverse effect. The range used is from 0.001 to 10. So, a higher learning rate results in rapid learning but there is weights oscillation, while lower learning rate leads to slower learning. Gradient descent is very slow if learning rate αis small and oscillates widely if α is very large. One efficient and common method that allows bigger learning rate without oscillations is through adding a momentum factor to normal gradient descent method.

The momentum factor is denoted by $\eta \in [0,1]$ and value of 0.9 is used for momentum factor. This approach is also useful when training data are different from majority of data. A momentum factor is used by either pattern by pattern updating or batch-mode updating. In batch mode, it effects a complete averaging over patterns. Even though averaging is only partial in pattern-by-pattern mode, it leaves useful information for weight updating.

Weight updating formulas used here are,

$$w_{jk}(t+1) = w_{jk}(t) + \underbrace{\alpha \delta_k z_j + \eta[w_{jk}(t) - w_{jk}(t-1)]}_{\Delta w_{jk}(t+1)}$$

and

$$w_{ij}(t+1) = w_{ij}(t) + \underbrace{\alpha \delta_j x_i + \eta[w_{ij}(t) - w_{ij}(t-1)]}_{\Delta w_{ij}(t+1)}$$

Momentum factor helps faster convergence.

where n is number of input neurons (equal to problem dimension), K number of hidden layer nodes, and function f is an activation function. A feed-forward MLPNN's structure is seen in Figure 1. W and θ are unknown weights to be learned. A common activation function is logistic function, with form:

$$f(x) = \frac{1}{1+e^{-cx}}$$

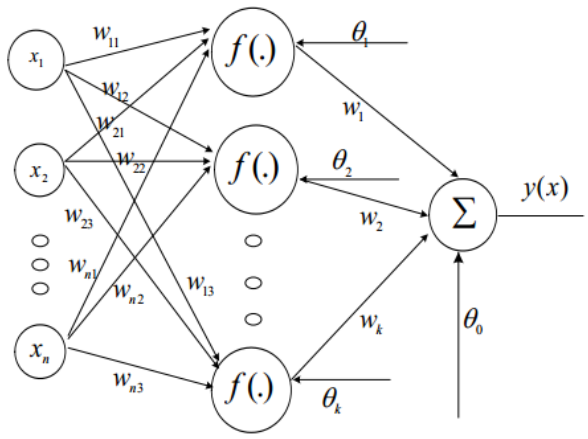

Fig.1. Feed Forward MLPNN Network Model

A. Proposed Activation

In this work an improved cubic spline based activation function is proposed. The new spline activation function reproduces whole cubic spline shape with directions specified by weight wj, j=1,..,n and written as:

$$\varphi\left(w_j x\right)\sum_{i=1}^{N} c_i \left|w_j x-\alpha_{ij}\right|^3$$

New activation function is written as:

$$f\left(x\right)=\sum_{j=1}^{n}\mu_j\varphi_j\left(w_j x\right)$$

μ_j and w_j are found using back propagation, thereby locating optimal set of parameters and coordinates.

Spline tracts are described through a coefficients combination. Activation function is represented by local spline basis functions controlled by 4 coefficients. Catmull-Rom cubic spline is used, and its ith tract expressed as:

$$F_i\left(u\right)=\begin{bmatrix}F_{x,i}\left(u\right)\\F_{y,i}\left(u\right)\end{bmatrix}=\frac{1}{2}\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}$$

When using evolutionary algorithms to train MLPNN, it is done through minimizing an error function. The latter function used is Mean Squared Error (MSE), which is calculated

$$MSE\left(w_{ij}\right)=\frac{1}{pm}\sum_{i=1}^{p}\sum_{j=1}^{m}\left(T_{ij}-F_{ij}\right)^2$$

where p denotes number of training patterns, m number of MLPNN outputs, T_{ij} target, F_{ij} actual value, both for output j and i input pattern. In nature inspired algorithms, the idea is that a solution in a population represents a connection weights vector of a MLPNN. Appropriate operators change weights, and the error by MLPNNs is used as fitness measure to guide selection.

The enhanced version of Glow-worm Swarm Optimization (GSO) algorithm is used to train MLPNN. GSO is based on a luciferin induced glow in a glow-worm which attracts mates or prey [21]. In GSO optimizing multi-modal functions, physical agents *i(i=1,..,n)* are randomly deployed in objective function space. An agent in a swarm decides movement direction by the strength of signal picked up from neighbours. The brighter the glow, the more is the attraction. GSO algorithm has 5 major steps to optimize a multi-modal function:

1) Each glow-worm *i* encodes objective function value $J(x_i(t))$ at its current location $x_i(t)$ into a luciferin value $l_i(t)$

$$l_i(t)=(1-\rho)l_i(t-1)+\gamma J\left(x_i(t)\right)$$

Where $l_i(t)$ represents luciferin level associated with glow-worm i at time t, ρ is luciferin decay constant (0 <ρ< 1), γ is luciferin enhancement constant, and $J(x_i(t))$ represents value of objective function at agent i's location at time t.

An initial random solution for a 3-2-1 Network is shown

0.314	0.872	0.545	0.76	0.213	0.302
0.498	0.231	0.169	0.609	0.683	0.184
0.881	0.564	0.085	0.5	0.093	0.271
0.314	0.924	0.678	0.111	0.838	0.422
0.032	0.589	0.166	0.912	0.899	0.158
0.817	0.168	0.429	0.622	0.07	0.744

The fitness computed using MSE as the parameter can be given by

0.736
0.359
0.717
0.763
0.138
0.741

2) Constructing neighborhood set *Ni(t)*

3) Each glow-worm i calculates move to j probability *pij(t)*

$$p_{ij}(t)=\frac{l_j(t)-l_i(t)}{\sum_{k\in N_i(t)} l_k(t)-l_i(t)}$$

where j ∈ Ni(t), Ni(t) = {j : dij(t) <rid (t); li(t) <lj (t)} is the set of neighbors of glow-worm i at time t, dij (t) represents Euclidean distance between glow-worms i and j at time t, and rid (t) represent variable neighbourhood range associated with glow-worm i at time t.

4) Select moving objects j* and calculate new location xi(t+1), s is a moving step

$$x_i(t+1)=x_i(t)+s*\left(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|}\right)$$

where xi(t) ∈ R^m is location of glow-worm i, at time t, in m-dimensional real space R^m, ||•|| represents Euclidean norm operator, and s (> 0) is step size.

The new solution based on the above is given by

0.893	0.914	0.449	0.049	0.734	0.459
0.109	0.021	0.453	0.903	0.161	0.425
0.816	0.654	0.049	0.057	0.167	0.672
0.337	0.055	0.27	0.874	0.208	0.013
0.125	0.669	0.339	0.696	0.493	0.504
0.893	0.179	0.39	0.552	0.095	0.592

5) Update radius of dynamic decision domain.

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta - |N_i(t)|\}\}$$

where β is a constant parameter and nt a parameter to control neighbours. The choice of parameters influences the algorithm's performance. Parameter values used are tabulated in table 1.

TABLE 1. The GSO parameters used

Parameter	Value
ρ	0.4
γ	0.6
β	0.08
n_t	5
s	0.03
l_0	5

Results and Discussion

The KC1 Dataset is used for the performance evaluation of the proposed technique; 2107 samples was used of which 1391 samples are used as training set and 716 samples are used for testing. The software complexity measures Base Halstead measures, LOC measure, Cyclomatic complexity, and Derived Halstead measures are used to classify the software modules. The proposed spline activation function reproduces the shape of whole cubic spline along the directions specified by weight wj, j=1,..,n. The MLPNN is made up of 20 input neurons and two hidden layers. Table 2 shows the results obtained from our experiments.

TABLE 2. Results obtained from our experiments

	MLP (Zheng et al.,)	**MLP with proposed activation**	**MLP with proposed GSO training**	**MLP with proposed activation & GSO training**
Classification accuracy	0.925	0.9335	0.9364	0.9497
Precision for defect	0.835	0.8619	0.8692	0.9043
Precision for no defect	0.9344	0.9415	0.944	0.9552
Recall for defect	0.5719	0.6199	0.637	0.7123
Recall for no defect	0.9818	0.984	0.9846	0.9879
F measure for defect	0.9575	0.9623	0.9639	0.9713
F measure for no defect	0.6788	0.7211	0.7352	0.7969

$$\text{Classification Accuracy } (\%) = \frac{(TN + TP)}{(TN + FN + FP + TP)}$$

$$\text{Precision } (\%) = \frac{TP}{(FP + TP)}$$

$$\text{Recall } (\%) = \frac{TP}{(FN + TP)}$$

where TP (True Positive) = Number of correct predictions that an instance is valid

TN (True Negative) = Number of correct predictions that an instance is invalid

FP (False Positive) = Number of incorrect predictions that an instance is valid

FN (False Negative) = Number of incorrect predictions that an instance is invalid

All the proposed techniques improves significantly over existing techniques in literature using Neural Network (Zheng et al.,). An improvement of over one percent in the classification accuracy reduces maintenance cost in software industry which is a recurring cost.

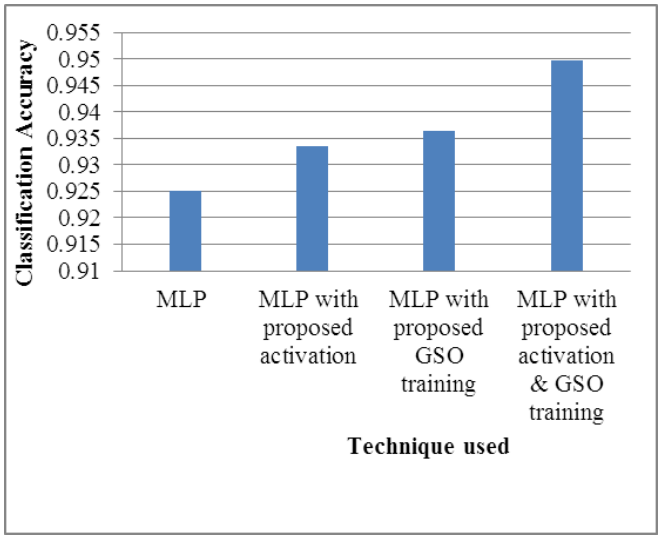

Fig.2. Classification accuracy

From figure 2, it is observed that the proposed MLP with proposed activation with GSO training increased classification accuracy by1.72% when compared with MLP with proposed activation.

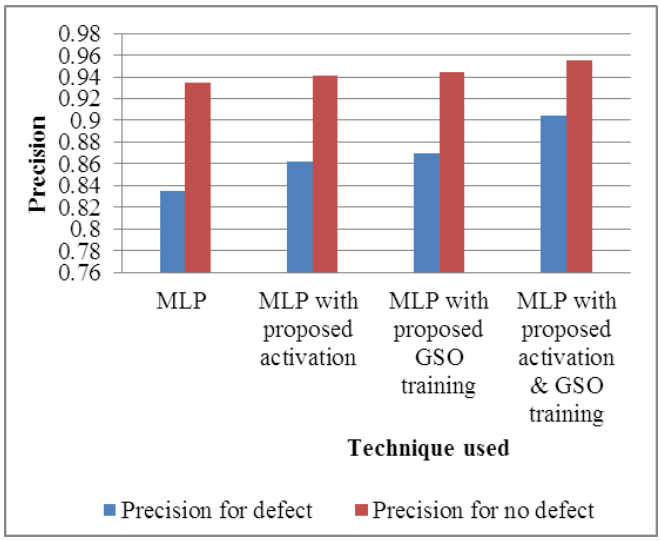

Fig.3. Precision

From figure 3, it is observed that the proposed MLP with proposed activation with GSO training increased precision by 4.80% and 1.44% with defect and no defect respectively when compared with MLP with proposed activation.

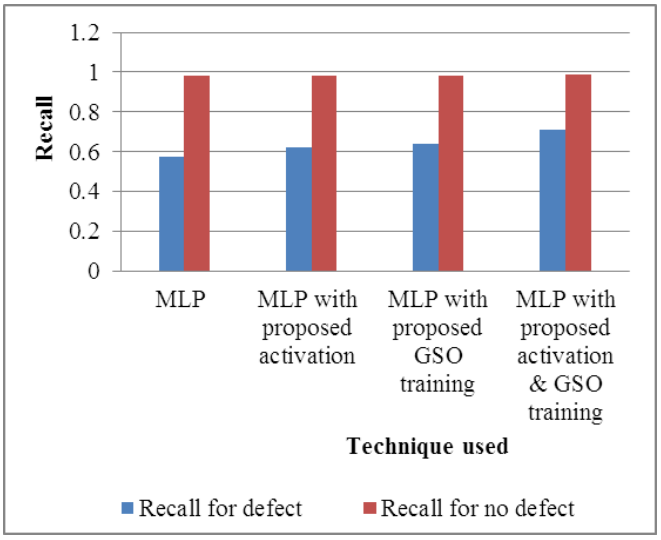

Fig.4. Recall

From figure 4, it is observed that the proposed MLP with proposed activation with GSO training increased recall by 13.87% and 0.39% with defect and no defect respectively when compared with MLP with proposed activation.

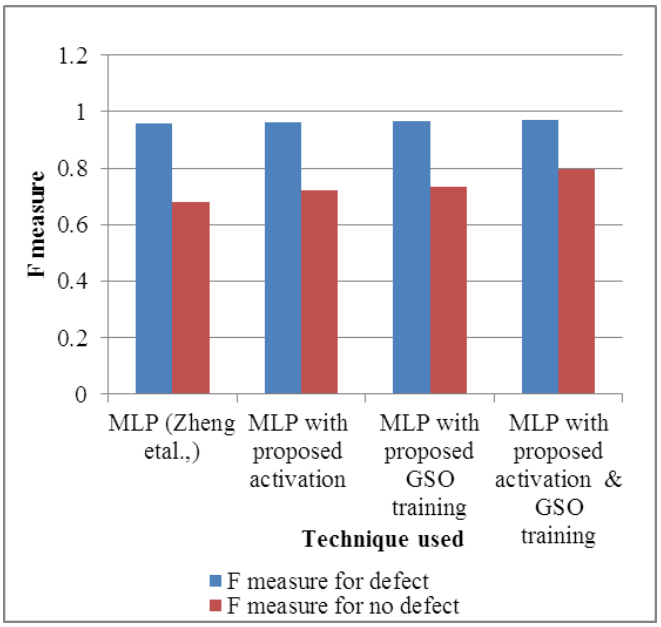

Fig.5. F measure

From figure 5, it is observed that the proposed MLP with proposed activation with GSO training increased f measureby0.93% and 9.98% with defect and no defect respectively when compared with MLP with proposed activation. From the results it can be seen that the proposed technique not only improves the classification accuracy but also improves both precision and recall.

Conclusion

Software defect prediction aims to improve software quality and testing efficiency by constructing predictive classification models from code attributes to ensure timely identification of fault-prone modules. SDP techniques assess systems dependability using defect metrics. Various techniques have been proposed in literature to improve the classification rate with Neural Network showing good performance. Since most work in literature used sigmoidal or tanh activation function, this work investigated a new activation function based on cubic splines. Neural Network suffer from parameter optimization problems which has not been addressed for the Software Defect Prediction Problem. To overcome the poor convergence of Genetic Algorithm, an improved Glow Swarm Optimization (GSO) algorithm to train the Neural Network parameter was proposed. The new MLPNN with cubic spline activation function and GSO training achieves classification accuracy of 94.97%. Further investigations can be carried out in the direction of hybridizing GSO algorithm with local search algorithms to improve the convergence characteristics.

References

[1] Koru, A., Liu, H., 2005. Building effective defect-prediction models in practice. IEEE Software, 23–29.

[2] Koru, A., Tian, J., 2003. An empirical comparison and characterization of high defect and high complexity modules. Journal of Systems and Software 67, 153–163.

[3] Emam, K., Benlarbi, S., Goel, N., Rai, S., 2001. Comparing case-based reasoning classifiers for predicting high risk software components. Journal of Systems and Software 55 (3), 301–310

[4] Catal, C., &Diri, B. (2009). A systematic review of software fault prediction studies. Expert systems with applications, 36(4), 7346-7354.

[5] Kkoshgotaar, T. M., &Seliya, N. (2004). Comparative assessment of software quality classification techniques: An empirical case study. Empirical Software Engineering, 9, 229–257.

[6] Thwin, M.M.T., Quah, T.S., Application of neural networks for software quality prediction using object-oriented metrics. Journal of Systems and Software, vol.76, no.2, pp. 147–156, 2005.

[7] Su, Y. S., & Huang, C. Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models.Journal of Systems and Software, 80(4), 606-615.

[8] Zheng, J. (2010). Cost-sensitive boosting neural networks for software defect prediction. Expert Systems with Applications, 37(6), 4537-4543.

[9] Hu, Q. P., Xie, M., Ng, S. H., &Levitin, G. (2007). Robust recurrent neural network modeling for software fault detection and correction prediction. Reliability Engineering & System Safety, 92(3), 332-340.

[10] Ma, C., Gu, G., & Zhao, J. (2011). Improved Neural Network based on Dynamic Predication Model of

Software Reliability. JCIT: Journal of Convergence Information Technology, 6(7), 349-357.

[11] Singh, Y., & Kumar, P. (2010). Application of feed-forward neural networks for software reliability prediction. ACM SIGSOFT Software Engineering Notes,35(5), 1-6.

[12] Okutan, A., &Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), 154-181.

[13] K. Mehrotra, C.K. Mohan and S. Ranka, "Elements of Artificial Neural Networks", Penram International Publishing, 1997

[14] Rumelhart, D.E., Williams, R.J., Hinton, G.E., Learning internal representations by error propagation, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1, 1986, pp. 318–362

[15] Tsai, J.T., Chou, J.H., Liu, T.K.: Tuning the Structure and Parameters of a Neural Network by Using Hybrid Taguchi-Genetic Algorithm. IEEE Transactions on Neural Networks, Volume 17, Issue 1, 2006, pp.69-80

[16] Mendes, R., Cortez, P., Rocha, M., Neves, J., Particle swarm for feedforward neural network training. In: Proceedings of the International Joint Conference on Neural Networks, Vol. 2, 2002, pp. 1895–1899

[17] Karaboga D., Akay B., Ozturk C., Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks, LNCS: Modeling Decisions for Artificial Intelligence, Volume 4617, 2007, pp.318-329

[18] Jayaraj, V., & Raman, N. S. (2013). Software Defect Prediction using Boosting Techniques. International Journal of Computer Applications, 65(13)

[19] Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Neural Networks, 1993., IEEE International Conference on (pp. 586-591). IEEE.

[20] Tamboli, Z. J., &Nikam, P. B. (2013). Study of Multilayer Perceptron Neural Network for Antenna Characteristics Analysis. International Journal, 3(8)

[21] Krishnanand, K. N., &Ghose, D. (2009). Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm intelligence, 3(2), 87-124