Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 5954432, 14 pages
https://doi.org/10.1155/2021/5954432

Research Article

Software Defect Prediction Based on Elman Neural Network and Cuckoo Search Algorithm

Kun Song,[1] ShengKai Lv,[1] Die Hu,[1] and Peng He [1,2]

[1]*School of Computer Science & Information Engineering, Hubei University, Wuhan 430062, China*
[2]*Hubei Key Laboratory of Applied Mathematics, Hubei University, Wuhan 430062, China*

Correspondence should be addressed to Peng He; penghe@hubu.edu.cn

Received 19 May 2021; Revised 6 October 2021; Accepted 13 October 2021; Published 22 November 2021

Academic Editor: Yun-Wen Feng

In software engineering, defect prediction is significantly important and challenging. The main task is to predict the defect proneness of the modules. It helps developers find bugs effectively and prioritize their testing efforts. At present, a lot of valuable researches have been done on this topic. However, few studies take into account the impact of time factors on the prediction results. Therefore, in this paper, we propose an improved Elman neural network model to enhance the adaptability of the defect prediction model to the time-varying characteristics. Specifically, we optimized the initial weights and thresholds of the Elman neural network by incorporating adaptive step size in the Cuckoo Search (CS) algorithm. We evaluated the proposed model on 7 projects collected from public PROMISE repositories. The results suggest that the contribution of the improved CS algorithm to Elman neural network model is prominent, and the prediction performance of our method is better than that of 5 baselines in terms of F-measure and Cliff's Delta values. The F-measure values are generally increased with a maximum growth rate of 49.5% for the POI project.

1. Introduction

With the increasing complexity of software and people's continuous demand for low cost, high quality, and maintainability of software in daily life, it is almost impossible to develop a software without any defects. As we know, defect is one of the key factors affecting software quality. It is essential to improve software quality before deployment, reduce system maintenance work, and detect and eliminate software defects early. Hence, software defect prediction is of high importance and an indispensable task.

Defect prediction techniques are often based on building models based on software metrics collected from similar projects or past releases. Such prediction models are used to classify the current project as defective or not defective. Previous research efforts to build accurate prediction models have been in either of the two following directions. The first one is the manual design of a set of specific software features to determine the defects, such as Halstead metrics [1] based on operand and operator counts, CK metrics [2] connected with function and inheritance counts, etc. The second one is code churn features [3] that contain the number of lines that are added/removed and the modified code, etc. With the increasing size of the codes, early manual investigations reduce the efficiency and accuracy of the review, while increasing the review cost.

An alternative approach is using machine-learning algorithms such as decision trees, support vector machines, Bayesian learning, and neural network. These techniques are well suited for real-life problems. Although they can process data that are imprecise, partially incorrect, or uncertain, the performance of these technologies varies on different datasets and parameters. Furthermore, in cases where the original dataset is not properly and effectively processed, it is difficult to get ideal results from the software prediction model [4]. Compared with other machine-learning algorithms, such as decision trees and Bayesian learning, neural network models are more dominant for data with higher dimension or larger amount of data. Therefore, this paper chooses to use the Elman neural network model.

Software products have a life cycle, and defects also have a time characteristic, such as the problem of too long average transaction response time, etc. In the existing studies, few researchers considered the effect of the time factor. To fill the gap, we propose an improved Elman neural network model, which optimizes the initial weights and thresholds using the improved Cuckoo Search (CS) algorithm.

The rest of this paper is organized as the following. The related work and background knowledge is presented in Sections 2 and 3. Then, in Section 4, we present the proposed neural network framework, followed by the experimental results in Section 5. Section 6 discusses potential threats to the effectiveness of our work. Finally, Section 7 concludes this paper and presents the directions for future work.

2. Related Work

There are many methods on defect prediction in the existing studies. For instance, in 2004, Kaner [5] defined metric estimation as the primary part of bug detection. Zhong et al. [6] also surveyed the cluster methods using k-means and Neural-Gas techniques and showed that the Neural-Gas method is more efficient in terms of Mean Squared Error (MSE). They also showed that the k-means method is faster than that of other existing methods.

Machine-learning (ML) techniques have also been widely used to estimate software faulty modules/classes [7]. Karim and Mahmoud [8] applied the Support Vector Machine (SVM) to predict defects and found out that the performance of SVM is better than that of the methods based on NASA datasets. Singh et al. [9] also compared the Decision Trees (DT) and Artificial Neural Networks (ANN) to predict faults of various severity levels.

The prediction performance of Back Propagation Neural Network (BPNN) for software defects was investigated by Paramshetti and Phalke [10]. In 2016, Al-Jamimi and Hamdi [11] validated the performance of the fuzzy-based models using real software project data. Madeyski and Kawalerowicz [12] also proposed the concept of continuous defect prediction. Felix and Lee [13] presented the application of an integrated machine-learning approach based on regression models constructed from these predictor variables, which enhanced the effectiveness of software development activities. In 2018, Huang and Strigini [14] applied the scientific understanding of human error mechanisms to predict software defects. The recent research works on software defects were summarized in Ref. [15] and the existing methods of defect classification were compared.

Qu et al. [16] also used a newly proposed network embedding technique and used automatically encoded class dependency relationships on low-dimensional vector space to improve software defect prediction, named node2defect. Tua and Danar Sunindyo [4] also added the process of selecting features using Rule Mining Association Methods (ARM) in the software defects prediction process and showed that using the Naive Bayesian (NB) method with ARM can improve the performance of the method using software metrics.

Ayon [17] proposed a method using Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) and then trained the model with different Neural Network methods. They then showed that their method achieves higher prediction performance compared with the general approaches.

Felix and Lee [18] focused on method-level defect prediction and constructed regression models to predict the estimated number of bugs. Paramshetti and Phalke [19] conducted a systematic study on machine-learning methods applied in software defect detection and provided a comparative study in the corresponding literature. Although various classification models have been proposed, the high-dimensionality of the dataset used for bug detection results in models with low accuracy. This is because the datasets with extreme features may have irrelevant and redundant features. Considering this issue, Malhotra and Khan [20] performed a comparison on nine open-source software systems written in Java using four mostly used feature extraction techniques.

Furthermore, in the latest research progress, Zhu et al. [21] proposed a probabilistic model to evaluate the most probable point (MPP) using cumulative distribution function of basic random variables; the results illustrate that the proposed model provides an efficient approach to obtain the MPP which is simpler and more accurate than the usual models. Zhu et al. [22] also proposed a hybrid iterative conjugate first-order reliability method (CFORM) and adaptive dynamical harmony search (ADHS) optimization, and they are developed for fuzzy reliability analysis (FRA) of stiffened panels. In 2021, they also compared the ability and accuracy of six heuristic algorithms based on social-inspired optimization in optimization of load-carrying capacities of HSS [23].

3. Preliminaries

3.1. Elman Neural Network. As the optimization of back-propagation (BP) network, the dynamic recurrent Elman neural network was proposed by J. L. Elman in 1990 [24]. Typically, the topology of the Elman neural network is divided into four layers: input layer, hidden layer, context layer, and output layer. The added context layer is used to store the preceding outputs of the hidden layer by using a positive feedback mechanism. This enables the model to adapt to the time changes. Figure 1 shows the basic structure of the Elman neural network.

The Elman neural network can effectively optimize the basic structure of the neural network. However, there exist some inherent drawbacks, e.g., it is hard to ascertain the number of input and hidden nodes. Such networks may converge to a locally optimal solution, and the fixed learning rate limits their convergence rate. Fortunately, several algorithms were proposed to address the above problems. For instance, a normalized risk-averting error criterion was proposed in literature [25] to avoid nonglobal local minima. Also, Ltaief et al. [26] proposed a fuzzy learning rate approach to adjust the convergence. Optimization of the hidden nodes by a genetic algorithm was also suggested in Ref. [27]. A method based on Genetic Algorithm (GA) and

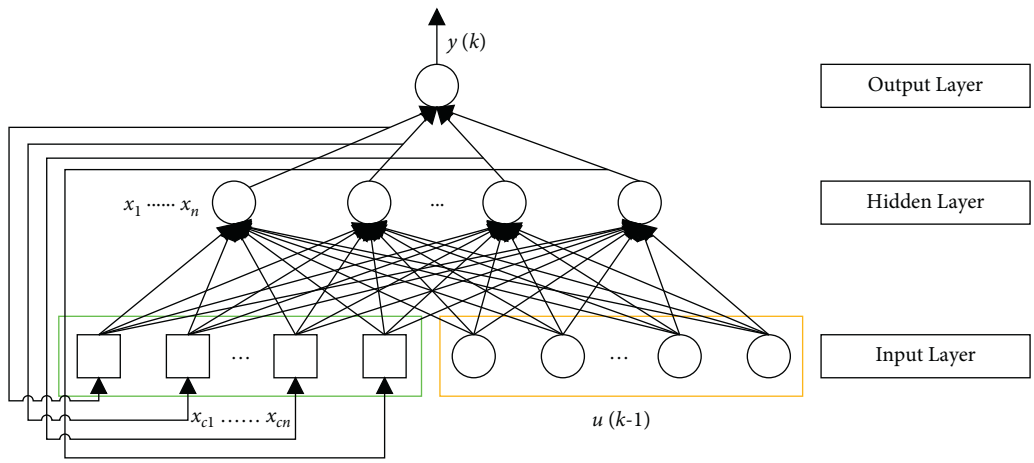

Figure 1: The structure of the Elman neural network model.

Particle Swarm Optimization (PSO) was also applied in several neural networks as in Ref. [17].

In this paper, we apply the principal component analysis (PCA) algorithm for data prepossessing to determine the input nodes. In addition, an improved Cuckoo Search algorithm is adopted to optimize the initial weights and thresholds of the Elman neural network. This can effectively improve the learning process and avoid trapping into local minima.

3.2. Principal Component Analysis. R. Bellman pointed out that the primary problem facing high-dimensional data analysis is the "curse of dimensionality" [28]. The idea of principal component analysis (PCA) was first proposed by K. Pearson in 1901 [29]. PCA is a classical dimension reduction measure that transforms a large dataset into a smaller set, and still obtains most of the information in the large dataset. The essence of this work was the Karhunen-Loeve transformation (K-L transformation for short) [30].

K-L transformation uses the minimum mean square error as the measurement criterion for data compression. K-L is the optimal orthogonal transformation in the sense of the minimum mean square error. Choubey et al. [31] explored classification techniques with PCA and PSO optimization. A range of feature reduction methods was also proposed recently to improve the performance of the neural network models for bug detection. In 2020, Malhotra and Khan [32] performed a comparison on nine open-source software systems using mostly used feature extraction techniques including the PCA algorithm.

Choosing appropriate input data is one of the effective strategies to establish the best Elman neural network model. It is evident that the irrelevant data not only lead to poor accuracy but also extend the training time. Therefore, before modeling in this paper, we will use PCA algorithm to reduce the dimension of the extracted data.

3.3. Cuckoo Search Algorithm. Based on the research of cuckoo's hatching behavior and Levy flight, Yang and Deb et al. proposed a meta-heuristic cuckoo search algorithm [33]. The main advantages of this algorithm are a small number of parameters, simple operation, easy implementation, random search path optimization, and strong optimization ability.

Wang et al. supposed that the choice of the probability of discovery P will affect the search for the optimal solution. For a large P, it is difficult for a better solution to converge to the optimal solution. Furthermore, a small P reduces the convergence speed. To address this issue, an adaptive parasitic-failure probability was introduced in Ref. [34]. Zhen et al. further showed that the Levy fly model lacks self-adaptability. To enable adapting to the dynamic adjustment of the interval between large and small steps and to coordinate the relationship between the accuracy and the global optimization capability [35]. A new Cuckoo Search Algorithm named CSM was proposed in 2020 by Ciftcioglu and Turkcan [36], which fast converges to the global extremum, meanwhile avoiding getting trapped in the local extremum.

The CS algorithm has a smaller number of parameters; also, it has a stronger optimization ability, which is different from traditional swarm intelligent optimization algorithms. It is also easier to be flexibly combined with other algorithms. To address the issues of slow convergence and incomplete global search, in this paper, we present an effective strategy based on combined adaptive parasitic-failure probability and step-size control. Such an algorithm is more reasonable and controls the step size in every stage of the CS algorithm, thus enhancing the efficiency of the basic CS algorithm.

4. Approach

In this section, we present an improved CS-Elman neural network model (CS-ENN model) for predicting software defects based on deep learning techniques. We first present the architecture of our optimized CS-Elman neural network model. Then, we elaborate on the components of our proposed model. The proposed CS-Elman neural network model is shown in Figure 2.

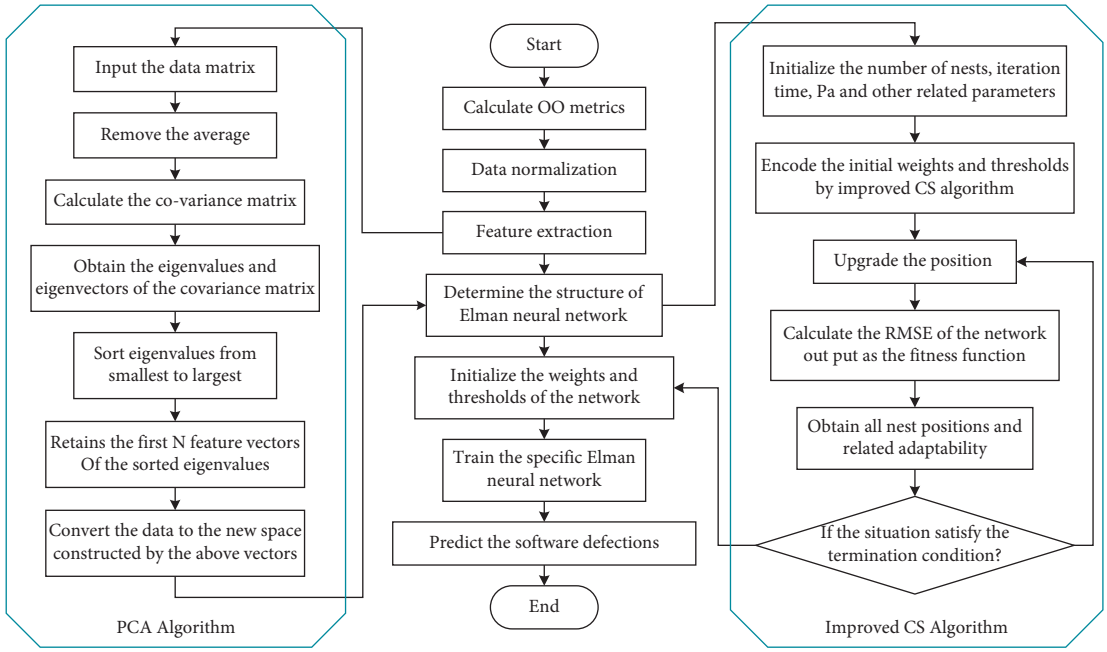

Figure 2: The overall structure of the CS-ENN model.

The left part in Figure 2 is the optimization of the input data of the Elman neural network; first remove the average value of the input data matrix, calculate the co-variance matrix, obtain the eigenvalues and eigenvectors of the co-variance matrix, sort the eigenvalues in ascending order, further retain the first 4 eigenvectors, and finally convert the data to the new space constructed by the above vectors. We also used the PCA algorithm to reduce the dimensionality of the OO metrics and extract the data containing the original feature information as the input of the neural network. The right part of the diagram in Figure 2 is the optimization of the Elman neural network, which includes an improved CS algorithm aiming to initialize and upgrade the weights and thresholds in the original network. The specific optimization process is introduced in detail in Section 3.3. After determining the structure of the neural network and initializing the weights and thresholds, train the target model, and finally use the model to predict software defects.

4.1. Basic Optimization of Elman Neural Network. The number of nodes at the input layer of the Elman neural network is determined by the dimension of source data. The number of nodes at the output layer is also ascertained by the research object. Using the PCA algorithm to reduce the dimension of the original data, we then classify the input layers into four parts, and the output layer is the prediction of the software defect. The performance of the neural network is also directly related to the number of hidden layer nodes [37]. Different numbers of hidden layers can cause different model prediction effects. Increasing the number of hidden layers may also reduce network errors and improve its accuracy, but it also increases the network complexity. This then increases the network training time and the tendency to overfit. Therefore, the choice of the number of hidden layer nodes is very important and has a great impact on the performance of the established neural network model.

It is generally believed that for a small number of hidden layer nodes, the network may be unable to establish complex judgment boundaries, hence might be unable to identify samples that have not been seen before. Furthermore, if the number of nodes is too large, the training time tends to be too long and the generalization ability of the network is also reduced and the error may also increase. Therefore, there exists an optimal number of hidden layer nodes. The basic principle of determining the number of hidden layer nodes is to use a compact structure under the premise of satisfying the accuracy requirements. This means taking the smallest number of hidden layer nodes. Here, we use the following empirical formula to determine the number of hidden layer nodes:

$$m = \sqrt{n + l} + a, \tag{1}$$

where m represents the number of hidden layer nodes, n denotes the number of input layer nodes, l is the number of output layer's nodes, and a is an integer between 1 and 10. According to (1), the number of hidden layer nodes is 5.

Elman neural networks are also affected by the learning rate which causes slow speed convergence. Generally, the learning rate is the speed at which the information accumulates in a neural network over time. It determines the speed at which the network reaches the optimal value or the speed at which the network parameters reach the optimal state for a specific desired output. In the plane graph of Stochastic Gradient Descent (i.e., SGD), the learning rate has nothing to do with the shape of the error gradient. This is because the global learning rate has nothing to do with the error gradient. If the learning rate is low, the training will become more reliable, but the optimization will take a long

time. This is because each step toward the minimum of the loss function is rather small. On the other hand, if the learning rate is high, the training might not converge or even diverge. The amount of the weight changes can be so large that the optimization crosses the minimum, making the loss function even worse.

Essentially, our goal is not to attenuate but to jump into the right place through attenuation. The learning rate must be selectively increased or decreased to achieve a global optimal value or the desired target value. Therefore, we proposed a dynamic approach to improve the convergence performance by adaptively adjusting the learning rate. The formula for adaptive adjustment of learning rate is:

$$\eta(t+1) = \begin{cases} (1+a)\eta(t), E_{t+1} < E_t, \\ (1+a)\eta(t), E_{t+1} > (1+b)E_t, \\ \eta(t), E_t < E_{t+1} \le (1+b)E_t, \end{cases} \tag{2}$$

where $\eta(t+1)$ and $\eta(t)$ represent the next and current learning rate of iteration, respectively. Also, E_{t+1} and E_t denote the next and current deviation of iteration, and a and b both present positive decimals. If the distance between the current and the last deviations is small enough, the algorithm will accelerate the learning speed to improve the convergence speed flexibly. If they have a significantly long distance, we should then reduce the learning rate and terminate the current operation to avoid deviating from the best solution. Therefore, we set the values of both a and b to 0.5.

In conclusion, depending on the specifics of the software project, after analyzing the feasible optimization of the parameters, we determine the final structure of the neural network, as presented in Table 1.

4.2. Features Extraction. The classification system is faster [2] where the input has fewer variables. Hence, we need to reduce the number of defective features. The PCA is used to reduce the number of input variables and to transform the original defect dataset to select the smallest subset of the data. We will briefly explain the PCA algorithm in the following.

First, we compute the mean, variance, and covariance matrix of the original dataset. Variance is used to measure the deviation between the random variables and their mathematical expectations. Covariance is used to measure the overall error of two variables reflecting the correlation between two sets of data. Variance is a special case of covariance, i.e., when two variables are the same. The mathematical formula is

$$V_i = \frac{1}{n-1} \sum_{m=1}^{n} \left(X_{im} - \overline{X_i}\right)^2,$$
$$C_{ij} = \frac{1}{n-1} \sum_{m=1}^{n} \left(X_{im} - \overline{X_i}\right)\left(X_{jm} - \overline{X_j}\right), \tag{3}$$

where v_i is the variance of variable i; C_{ij} is the covariance of variable i and variable j; X_{im}, X_{jm} represents the value of the variables i and j in sample m, respectively; and $\overline{X_i}$, $\overline{X_j}$ mean the mean value of the variables i and j, respectively.

Table 1: Parameters determination of the network structure.

Parameter	Optimal value
Input nodes	4
Hidden nodes	5
Output nodes	1
Training time	800
Learning rate	0.1
Target error accuracy	0.001

The next step is to calculate the eigenvalues of the covariance matrix and the corresponding eigenvectors. The eigenvalue is expressed as:

$$\mathrm{Av} = \lambda v, \tag{4}$$

where A is a square matrix of order n, and v is a nonzero n-dimensional column vector. There is a number v that makes this equation true, then λ is one of the eigenvalues of A, and the vector v corresponding to this value is called the eigenvector of A.

We then calculate the principal component (factor) based on the characteristic value and cumulative variable (%). At this point, the principal component is less than or equal to the number of original variables:

$$Y_i = v_{i1}x_1 + v_{i2}x_2 + \cdots + v_{in}x_n, i = 1, 2, \ldots, n, \tag{5}$$

where Y_i donates the ith principal factor, x_1 is the original variable, n is the number of original variables, and $(v_{i1}, v_{i2}, \ldots, v_{in})'$ is the eigenvector of the correlation matrix of the original defect dataset.

In the principal component space, the variance of the first principal component to the original data is the largest, and the largest variance represents the largest amount of information in the original data. The variance of each subsequent component is, therefore, lower than the subsequent component. In practice, the variance contribution rate of the principal component is often used to reflect the content of the related original information. The variance contribution rate R_k and the accumulative contribution rate A_k are

$$R_k = \frac{v_k}{\sum_{i=1}^{p} v_i},$$
$$A_k = \sum_{i=1}^{k} R_i, \tag{6}$$

where v_k means the kth eigenvalue of the covariance matrix of the original data. Generally, we just select the first k principal components according to the calculated A_k above and the value is larger than 85% in common.

The PCA algorithm only needs to retain the eigenvector group matrix A and the mean vector v of the samples and can project new samples into a low-dimensional space through simple vector subtraction and matrix-vector multiplication. The computational cost of algorithms is also an

important motivation for dimension reduction. We also note that the data are affected by noise; hence, the eigenvectors corresponding to the smallest eigenvalues are often related to the noise. Therefore, PCA has also a denoising effect. Therefore, to reduce the dimension of the complicated original data and construct the optimal defect-recognition model based on the obtained accumulative contribution rate, here we obtain the most suitable subsets of the principal components.

After dimension reduction using the PCA algorithm, we obtain four principal components. These components represent the characteristic of the four-dimensional indicators of the code. We then input these four variables into the neural network to train the model to predict the software defect.

4.3. Improved Cuckoo Search Algorithm. Elman neural network uses the gradient descent method to find the corresponding optimal weights and thresholds. Therefore, it may easily locally converge, and its convergence speed is low. To address these shortcomings of the Elman neural network, the Cuckoo search algorithm (i.e., CS algorithm) is introduced into the traditional network prediction model. The search algorithm optimizes its weights and thresholds to improve the prediction efficiency of the software defect.

4.3.1. Traditional Cuckoo Search Algorithm. The CS algorithm is an intelligent heuristic algorithm that combines the cuckoo breeding method with the Levy flight search principle. This algorithm finds the global optimal solution. Compared with the GA and PSO algorithms, the CS algorithm has better generality performance and fewer parameters (including the probability $P(a)$ and population size n parameters found in birds and eggs). There is a good balance between the global collection and local search strategy in the CS algorithm, which makes it more efficient.

Cuckoos have an aggressive breeding strategy. Some cuckoos lay their eggs in other nests and remove the nest owners' eggs. Some cuckoos also lay the same colors and patterns as the nest owners' eggs. These behaviors have improved the hatching rate of their eggs. In cases where the nest owner finds foreign eggs in the nest, it throws them away or abandons the nest to rebuild its nest elsewhere. Cuckoos also choose those nests in which there are recently laid eggs to lay their eggs. Generally, the incubation time of cuckoo's eggs is shorter than that of the nest's owner. Once the first cuckoo hatches, the instinctive action is to push down other eggs in the nest or imitate the call of the nest owner. This increases the food supply and improves its survival rate.

Various studies have shown that the flight behavior of many animals and insects exhibits the power law of Levy flight characteristics [38]. For instance, the fruit flies are accompanied by a rapid 90° turn from time to time in a series of straight flight paths. This pattern results in a Levy flight-like intermittent and irregular search pattern. Levy flight can be described as a moving entity that can occasionally take unusually large steps. To change the behavior of a system, the movement direction is random. The length of the movement step is also distributed according to the power rate. Various studies have shown that Levy Flight has a good performance for solving optimization problems and performing optimization searches [39].

The cuckoo search algorithm has three ideal rules:

(1) Each cuckoo lays only one egg at a time, and randomly selects the nest to place its egg,

(2) The best nest (solution) with the best-quality eggs will be retained for the next generation,

(3) The available nest master bird's n is fixed, and the nest master bird finds the foreign cuckoo eggs with the probability of Pa $\in (0, 1)$. In such cases, the nest owner discards the foreign eggs or leaves the old nest and builds a new nest.

Given the above three conditions, the update formula of the bird's nest position is

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus L(\lambda), \quad i = 1, 2, \ldots, n, \tag{7}$$

where $x_i^{(t+1)}$ is the updated position of the nest in the tth generation, and α donates the step size that is drawn from a normal distribution. The product $\oplus$ represents entry-wise multiplications and $L(\lambda)$ denotes the search path of the random walk via Levy flight. The random length of step s follows Levy distribution:

$$L(s, \lambda) \sim s^{-\lambda}. \tag{8}$$

4.3.2. Adaptive Parasitic Failure Probability and Step-Size Control Optimization. The probability of parasitic failure is represented by P_a. In the standard CS algorithm, P_a remains unchanged. This means that in the iterative process of the CS algorithm, whether it is a better bird's nest position or a poor bird's nest position, the parasitic failure occurs with the same probability P_a. If P_a is relatively large, the better bird's nest position is easy to be replaced and difficult to retain. Also, it is hard to converge to the optimal solution; on the contrary, the poor bird's nest position is not easy to be replaced, this causes a slower convergence.

To prevent the above situation, in the initial iteration of the iteration process, the CS algorithm should receive new solutions with a greater probability, speed up the convergence speed, and keep small P_a in the later iteration to retain better solutions. Therefore, we propose an adaptive parasitic failure probability P_a, as

$$P_a = P_{\min} + (P_{\max} - P_{\min}) \times \left(1 - \frac{t}{T}\right)^m, \tag{9}$$

where the parasitic failure probability ranges between $P_{\min}$ and $P_{\max}$; t and Tdenote the current and maximum number of iteration, respectively. In equation (9), m is a positive nonlinear factor to control P_a's declining rate. For $m = 1$, P_a is linearly decreased, which means the value m should not be set beyond 1. In our experiments, $P_{\min}$ and $P_{\max}$ are set to 0.1 and 0.5, respectively, and m is assumed to be 0.5. The total

time of iteration is also set to 500 to adaptively modify the parasitic failure probability during the iteration.

Levy flight path is a random process. In the course of the flight there exist high-frequency short steps and low-frequency long steps. Therefore, the search for a global solution indicates strong random jumps. Therefore, the algorithm's global optimization ability is strong, but also leads to the algorithm search for the location near the nest appear incomplete. When the number of iterations is relatively high, the nest locations that contain local information are not effectively used. This results in low convergence accuracy, and it becomes difficult to converge to the optimal solution.

During the flight, the high-frequency short step and the low-frequency long step alternately happen. Therefore, the Cuckoo search algorithm is very strong when searching the global solution space. The random jumps improve the algorithm's global optimization ability, but it also causes the algorithm to appear incomplete when searching for the location near a certain nest. When the number of iterations is relatively high, the better nest positions may be lost due to the long steps. Therefore, the local information of these nest positions is not effectively used. This results in a low convergence accuracy and difficulty in converging to the optimal solution. For this reason, we propose an adaptive step-length control, so that the CS algorithm can effectively control the step-length in each stage of the iterations. This enables the algorithm to retain an excellent position solution. The adaptive step-length control formulation is defined as follows:

$$\alpha_i = \frac{\alpha_{\max} + \alpha_{\min}}{2} \cdot \frac{\left|x_i - \overline{x}\right|}{\beta_i}, \qquad \beta_i = \frac{\sum_{j-1}^{n-1}\left(x_i - x_j\right)}{n-1}, \tag{10}$$

where α_i presents the next step's size of x_i, $\alpha_{\min}$ and $\alpha_{\max}$ denote the minimum and maximum step sizes, respectively. Here, we set $\alpha_{\min}$ and $\alpha_{\max}$ to 0.5 and 1.5, respectively, x_i is the solution of the current nest, $\overline{x}$ is the average of all current nest position solutions, and β_i is the average difference between the current and other nest position solutions. For a large distance between the current and average nest position solutions, the step size is increased by increasing α_i. Conversely, when the distance becomes smaller, it is decreased as the step size is also decreased. This method effectively utilizes the information of the local solution, avoids the excessive jump randomness of the traditional CS algorithm, and greatly improves the searchability of the optimal solution of the original CS algorithm.

4.3.3. Optimization of Network Weights and Thresholds Using the Improved Cs Algorithm. The objective is to address the issue of slow convergence of the neural network and converging to a local optimum when initializing the weights and thresholds of the Elman neural network. Here, we apply the improved Cuckoo search algorithm and further use the Elman neural network as the fitness function of the metaheuristic algorithm.

The procedure of the improved Elman neural network using an improved CS algorithm (CS-ENN) is shown in Figure 3.

After determining and initializing the weights and thresholds of the basic Elman neural network, we use the root mean square error (RMSE) as the fitness function to find the best initial weight and threshold of the Elman neural network of each iteration. The improved search algorithm continues to iterate until it reaches the convergence condition. The convergence condition is that the maximum number of iterations is reached, or the minimum root mean square error is met.

5. Experiments

In the previous section, we introduce the entire process of modeling and implementation of the CS-ENN predictive model. The model is also proved to be theoretically feasible. Here, we apply our theoretical model to real data for verification. To obtain more accurate and complete data and make the experiment credible, we use the basic defect detection indicators obtained by the general 7 different versions of Java projects. This paper conducts experiments using the prediction model and basic data proposed above and analyzes the results accordingly. Considering that the entire software defect detection process includes several specific processing procedures, it is important to ensure that we use appropriate and effective methods at each step. Figure 4 shows the specific prediction processing flow.

5.1. Dataset and Experimental Setup. To facilitate replication and validation of our experiments, we use publicly available data in the PROMISE data repository. The dataset contains 7 projects and 27 versions. Table 2 presents the basic description of these projects.

As it is seen, the average number of files for the project ranges from 150 to 830, and the minimum defect rate for the project is 18.87% and the maximum is 38.65%. Clearly, each open-source project has a series of versions, due to the iterative update during the development process.

The data refer to 20 metrics, include 11 CK metrics and its variants [2], 2 Martins metrics [40], 5 QMOOM metrics, and 2 McCabe's CC metrics [41]. Table 3 presents the detailed introduction of the metrics.

As shown in Table 2, this is an important feature that distinguishes this application from other statistical analyses. Due to the complexity of the data, they cannot directly reflect their temporal characteristics. Therefore, we slice the existing data with version granularity, and add the label field representing the time factor. If there are bugs in the previous version, the field value is 1, otherwise 0.

Having many indicators greatly increases the complexity of the analysis problem. To address this issue, we use the PCA. We use the SPSS software to perform the PCA algorithm and obtain the correlation coefficient matrix (see Table 4).

From the table, the correlation coefficient between wmc and rfc is about 0.869, and the figure between max_cc and avg_cc is calculated as 0.772. The results in the Table indicate

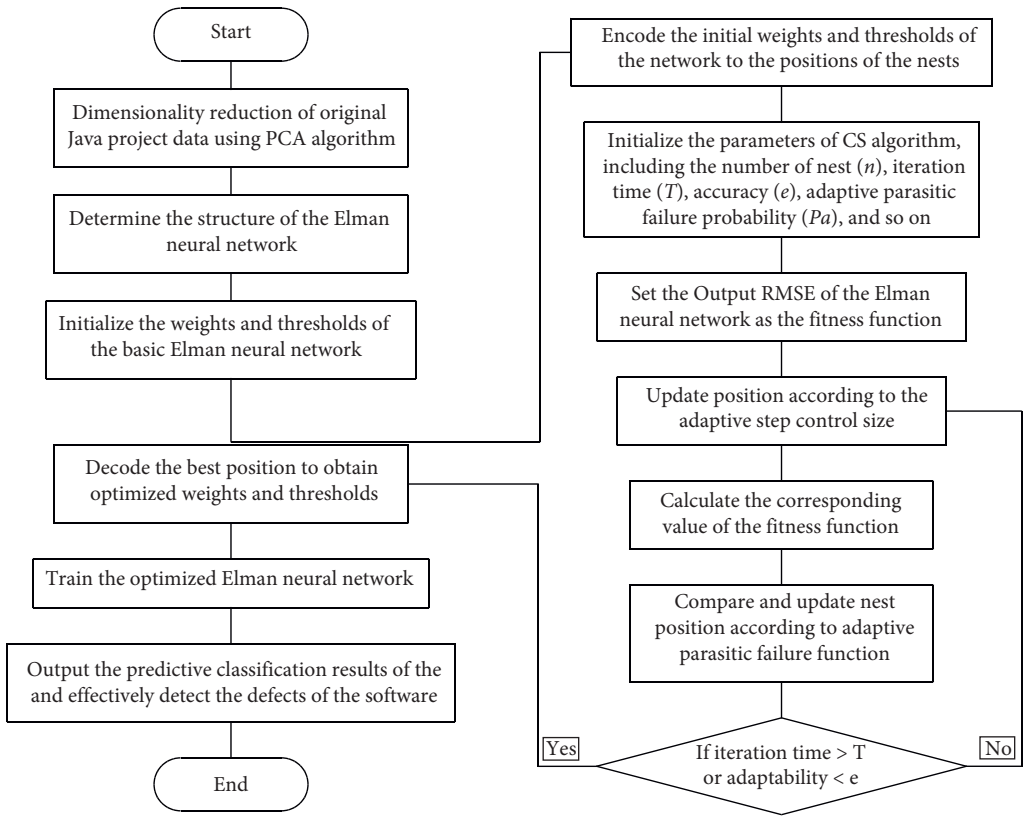

Figure 3: The prediction process of the CS-ENN model.

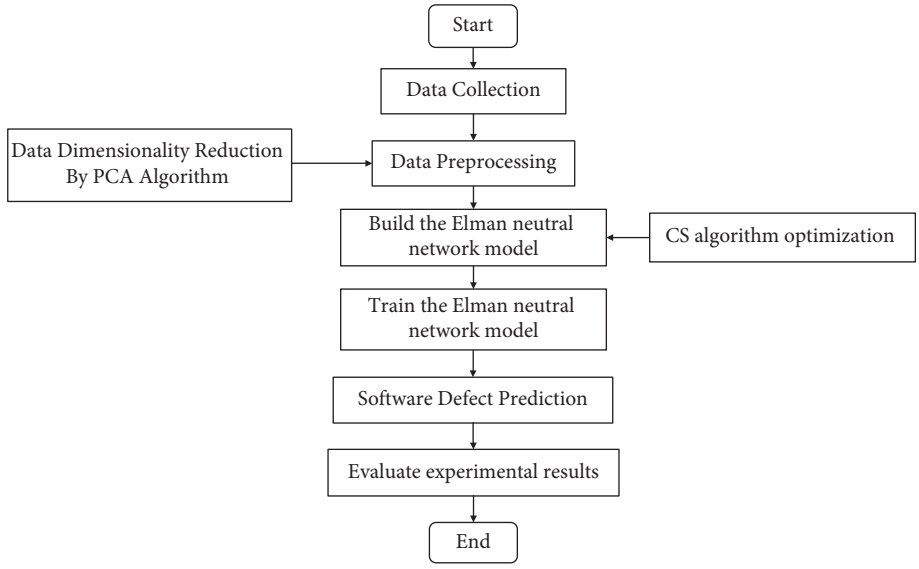

Figure 4: The diagram of the prediction steps.

that there are certain correlations between the influencing factors. Therefore, it is necessary to use PCA to eliminate the correlation between indicators when using neural networks to predict software defects. According to the analysis and calculation above, we finally obtain 4 new principal components which are then input to the Elman neural network model.

On the acquired dataset, we notice that if the processed data are directly used as the training sample of the prediction model, the prediction result error is often large, and

Table 2: The statistics of 7 projects.

Project	Description	Release	Avg files	Avg defect rate (%)
Ant	Java-based build tool	1.3, 1.4, 1.5, 1.6, 1.7	340	19.58
Camel	Enterprise integration framework	1.0, 1.2, 1.4, 1.6	696	18.87
log4j	Logging library for java	1.0, 1.1, 1.2	150	50.44
poi	Java library to access Microsoft format files	1.5, 2.0RC1, 2.5.1, 3.0	345	49.82
Velocity	A template engine based on java	1.4, 1.5, 1.6.1	213	58.47
Xalan	A library for transforming library	2.4.0, 2.5.0, 2.6.0, 2.7.0	830	52.14
Xerces	XML parser	1.2.0, 1.3.0, 1.4.4, init	411	38.30

Table 3: Description of object-oriented metrics.

Variable	Description
CK suite (6)	
wmc	Weighted methods per class
Dic	Depth of inheritance tree
Noc	Number of children
Cbo	Coupling between object classes
rfc	Response for a class
Lcom	Lack of cohesion in methods
Martins metrics (2)	
Ca	Afferent couplings
Ce	Efferent couplings
QMOOM suite (5)	
npm	Number of public methods
dam	Data access metric
Moa	Measure of functional abstraction
Mfa	Measure of aggregation
cam	Cohesion among methods
Extended CK suite (5)	
Ic	Inheritance coupling
Cbm	Coupling between methods
Amc	Average method complexity
lcom3	Normalized version of LCON
McCabe's CC (2)	
max_cc	Maximum values of methods in the same class
avg_cc	Mean values of methods in the same class
Loc	Lines of code
Bug	Nonbuggy or buggy

Table 4: Correlation coefficient matrix among influencing metrics.

Metric	wmc	Cbo	rfc	...	max_cc	avg_cc
wmc	1	0.483	0.868	...	0.437	0.32
Cbo	0.483	1	0.408	...	−0.019	−0.038
rfc	0.868	0.408	1	...	0.358	0.311
...	...	...	...	...	...	...
max_cc	0.437	−0.019	0.358	...	1	0.772
avg_cc	0.32	−0.038	0.311	...	0.772	1

saturation also occurs. Therefore, we normalize the processed data to avoid the impact of unprocessed data on the prediction performance. Furthermore, considering that the scale of the metrics is so large, we apply the PCA algorithm to reduce the dimensions of the original data to 4 indicators without losing the characteristics of the data. The data are then used as the input of the Elman neural network for subsequent experiments.

5.2. Evaluation Measures. To measure the performance of defect prediction, we use the following metrics: Precision, Recall, F-measure, and Cliff's Delta. The first three metrics are widely adopted to evaluate defect prediction techniques [41, 42]. All the four metrics are introduced below.

The value of Precision is the ratio of the number of correct predictions to the total number of positive predictions which represents how many predictions are accurate. The calculation formula is as follows:

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}, \tag{11}$$

where true positive records the number of predicted buggy files that truly contain bugs, while false positive is the number of predicted buggy files that are not buggy. The value of false negative is the number of predicted nonbuggy files that are truly buggy.

Recall determines the number of positive pictures predicted to be positive pictures in all annotated pictures, which indicates how many have been recalled from the perspective of labeling. The calculation formula is as follows:

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}. \tag{12}$$

As is seen in (11) and (12), the Precision and the Recall values are a pair of contradictory indicators. In the actual model evaluation, it is incomplete to evaluate the model only with the most common indicators, Precision or Recall.

However, it is inevitable to use these two values of Precision and Recall when evaluating the classify model. Therefore, we add the F-measure to the evaluation measures, which is defined as

$$\text{F} - \text{measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{13}$$

which is the weighted harmonic average of Precision and Recall.

The Cliff's Delta statistic is a nonparametric effect size measure that quantifies the difference between the two sets of observations except for the p-value interpretation. Such measurement of dominance reflects on the degree of overlap between two distributions. This measurement can be recognized as a useful supplementary analysis of the corresponding hypothesis test [42]. The Cliff's Delta estimator is defined as:

$$\text{Delta} = \frac{\#(x_1 > x_2) - \#(x_1 < x_2)}{n_1 n_2}, \quad (14)$$

where x_1 and x_2 are scores of approaches 1 and 2, respectively. In (14), n_1 and n_2 mean the sizes of the sample groups of the above approaches. The cardinality symbol # denotes counting. This statistic measures the probability that the score of a sample selected from one group is greater than that selected from the other group, minus the reverse probability. The value of Cliff's Delta metric is in the range from −1 to 1. A value of 0 means that the two group distributions fully overlap, while the size of −1 or −1 represents that there is no overlap between the two distributions.

To evaluate the performance of our proposed model in defect prediction, we compare the CS-Elman neural network model with the conventional defect prediction approaches. In this study, our baselines of the conventional methods consist of 6 machine-learning techniques, which are shown in Table 5.

5.3. Experiment Results. In this section, our experiment is divided into the following three parts. First, we apply the Elman neural network and BP neural network to predict the software defects. For the prediction of data with time factors, we use Cliff's Delta indicator to verify whether the Elman neural network is better than that of the general neural network. Second, we use CS-Elman neural network, Elman neural network, CS-BP neural network, and BP neural network to conduct experiments to verify whether the CS algorithm can optimize neural networks (including Elman neural network and ordinary neural network). Finally, we compare the CS-Elman neural network with the baselines, to verify whether the classification of the neural network method performs better.

5.3.1. The Efficiency of Cs-Elman Neural Model Compared with the Basic Neural Network Model. Figure 5 indicates that the F-measure of models based on Elman neural network is higher than that of the BP neural network. The result also shows that Elman neural network optimized by the CS algorithm has a slight advantage, the average value of the former is 0.4563, while the latter is 0.4212. Therefore, the CS-Elman neural network outperforms the two basic neural networks in our context.

5.3.2. The Impact of Cs Algorithm on Neural Network Model. In order to further explore whether using the improved CS algorithm to optimize the Elman neural network will have more advantages than other algorithms, we carried out a comparative experiment. Figures 6 and 7 show the results of our experiments.

As is seen, the F-measure values of CS-Elman neural network are mostly higher than that of the basic neural network. The average F-measure values of the optimized CS-Elman neural network model and CS-BP neural network model are 0.4562 and 0.4084, respectively, whereas the values of the unoptimized models are 0.4212 and 0.3228, respectively. Furthermore, the GA-PSO improved networks outperform the basic models. It is however less significant while using the CS algorithm.

In addition, we obtain the Cliff values of these models, to more effectively verify the significant influence of the CS algorithm on the neural network model. The calculation results are shown in Table 6.

In Table 6, Cliff's Delta value between CS-Elman neural network and Elman neural network model is 0.1837, and the value between CS-BP neural network and BP neural network model is 0.3961. These indicate that the CS algorithm does effectively improve the performance of the neural network model on software defect prediction by initializing the best weights and thresholds. Moreover, these values of Cliff's Delta are both positive, which means that the CS algorithm has a significant positive influence on these two kinds of neural networks.

5.3.3. The Advantage of Cs-Enn Compared with the Baselines. We compared the performance of software defect prediction between the improved Elman neural network classifier and the general classifiers. Also, we introduce the state-of -the art method of Node2Defect to verify the performance of CS-Elman neural network. Here, we investigated whether the classification effect of the CS-Elman neural network is better than that of the commonly used classifiers.

As shown in Figure 8, in the *camel* project, the performance of the Naive Bayes approach for detecting bugs is slightly better than that of our model. In the *velocity* project, all models show similar performance. The F-measure value of our CS-Elman neural network is generally higher than that of the other classifiers including the newly proposed Node2defect method. Therefore, we can conclude from the figure that our model has a better detection effect than other basic classifiers and can better detect bugs in the software.

Next, to test whether our model is significantly different from the general classifiers, here we obtain the Cliff's Delta values of these classification models. The specific results are presented in Table 7.

The above results confirm that our proposed improved Elman neural network classifier has a certain significant difference with the baselines. Also, because all the values of Cliff's Delta are positive, it can be concluded that our model has a positive difference compared with the general classifier. The results also indicate that our model has a better effect on software defect detection.

6. Threats to Validity

In this study, we obtain several significant results to examine the proposed network model. However, potential threats to the validity of our work remain valid.

Threats to construct validity are concerned with the relationship between theory and observation. These threats have a main connection with the static code we used. The artificial errors that occurred in the project code may inevitably have a repercussion in the experiments. However, these datasets obtained from the commonly used PROMISE

Table 5: Description of baselines.

	Baselines	Description
LR	Logistic regression	LR is a machine-learning method used to solve two classification (0 or 1) problems based on statistical.
NB	Naive Bayes	Naive Bayes classifier is a probabilistic classifier based on the use of Bayes' theorem under the assumption of strong independence between features.
BN	Bayesian networks	NBC is an uncertain causal reasoning model, which can make classification reasoning from incomplete, inaccurate, or uncertain information.
J48	C4.5 decision tree	C4.5 decision tree is based on the original decision tree that each internal node represents a judgment on an attribute, each branch represents the output of a judgment result, and finally each leaf node represents a classification result.
BP	Back propagation neural network	BP is a multi-layer feedforward neural network trained according to the error back-propagation algorithm, and is the most widely used neural network, which can be applied to classification.
N2D	Node2Defect	Node2Defect is a newly presented network embedding technique, which can learn to encode dependency network structure into low-dimensional vector spaces automatically, and improves the performance of software defect prediction.

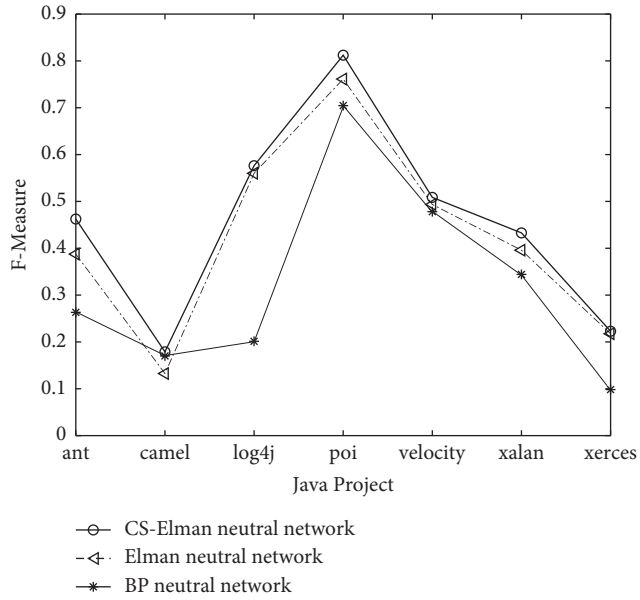

Figure 5: CS-ENN, ENN, and BP prediction results.

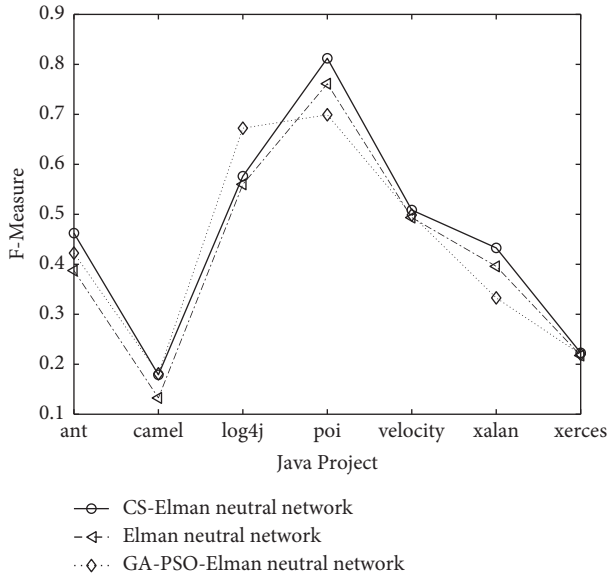

Figure 6: CS-ENN and ENN prediction results.

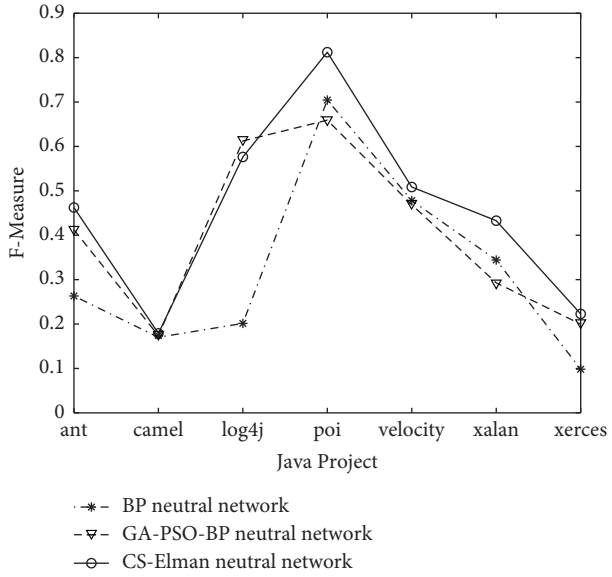

Figure 7: CS-BP and BP prediction results.

Table 6: The Cliff's Delta of the 4 networks.

Cliff' delta	CS-Elman neural network	Elman neural network	CS-BP neural network	BP neural network
CS-Elman neural network	—	0.1837	0.2245	0.3878
Elman neural network	−0.1837	—	0.0816	0.3469
CS-BP neural network	−0.2245	−0.0816	—	0.3961
BP neural network	−0.3878	−0.3469	−0.3961	—

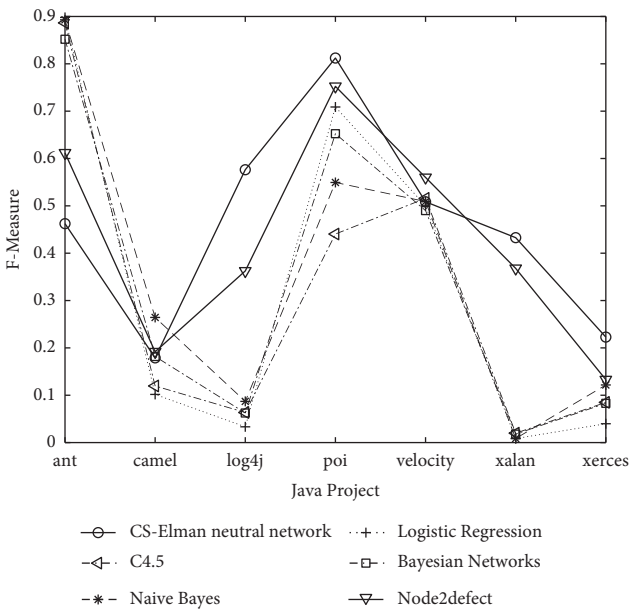

Figure 8: CS-ENN and baseline prediction results.

repository have been validated and applied to several prior studies. Therefore, we believe that our results are not only suitable for our collected datasets in the specific repository but also credible for other open-source projects.

Threats to the external validity are concerned with the generalization of the results of experiments. The primary threat could be associated with the single source of data from the PROMISE repository. However, we selected 7 different projects to examine the model. Also, the resembling trends have been shown with other language software datasets in previous studies. Although we cannot completely control the threats to external effectiveness, in an internally valid study, we can prove that the effect of treatment under specific research conditions is valid. Therefore, our model can be considered for other datasets.

Table 7: The Cliff's Delta of baselines with CS-Elman neural network.

Cliff' delta	LR	NB	BN	J48	N2D
CS-elman neural network	0.3061	0.2245	0.3877	0.3878	0.2978

7. Conclusion

In this paper, to consider the time characteristics of software defect, we proposed a novel predictor based on Elman neural network optimized by an improved CS algorithm. Specifically, we optimized the initial weights and thresholds of the Elman neural network by incorporating adaptive step size in the traditional CS algorithm. We evaluated the predictor on 7 Java projects, and the results confirmed that the Elman neural network model outperforms the general neural network model. In addition, our CS-ENN algorithm is also better than the general prediction techniques in terms of

F-measure values and Cliff's Delta values. The F-measure values are generally increased with a maximum growth rate of 49.5%.

In future work, on the one hand, we plan to validate the generalization of our model with more projects written in different languages. On the other hand, we can employ our model to predict the level of severity of software defect related to security. This enables efficient discovery of defects and solving the practical issues in software development. Last but not least, we also plan to discuss the possibility of considering not only ENN model but also CNN, RNN, and graph neural networks for network embedding, respectively.

Data Availability

We use publicly available data in the PROMISE data repository. http://promise.site.uottawa.ca/SERepository/datasets-page.html.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. H. Halstead, *Elements of Software Science (Operating and Programming System Series)*, North-Holland, New York, NY, USA, 1977.

[2] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[3] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 279–289, Silicon Valley, CA, USA, November 2013.

[4] F. M. Tua and W. Danar Sunindyo, "Software defect prediction using software metrics with naïve Bayes and rule mining association methods," in *Proceedings of the 2019 5th International Conference on Science and Technology (ICST)*, vol. 1, pp. 1–5, Yogyakarta, Indonesia, July 2019.

[5] C. Kaner, "Software engineering metrics: what do they measure and how do we know?" in *Proceedings of the IEEE International Software Metrics Symposium*, pp. 1–12, Chicago, IL, USA, September 2005.

[6] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *IEEE Intelligent Systems*, vol. 19, no. 2, pp. 20–27, 2004.

[7] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[8] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[9] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Software Quality Journal*, vol. 18, no. 1, pp. 3–35, 2009.

[10] P. Paramshetti and D. Phalke, "Survey on software defect prediction using machine learning techniques," *International Journal of Scientific Research*, vol. 3, no. 12, pp. 1394–1397, 2014.

[11] H. A. Al-Jamimi, "Toward comprehensible software defect prediction models using fuzzy logic," in *Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 127–130, Copenhagen, Denmark, August 2016.

[12] L. Madeyski and M. Kawalerowicz, "Continuous defect prediction: the idea and a related dataset," in *Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 515–518, Madrid, Spain, May 2017.

[13] E. A. Felix and S. P. Lee, "Integrated approach to software defect prediction," *IEEE Access*, vol. 5, pp. 21524–21547, 2017.

[14] F. Huang and L. Strigini, "Predicting software defects based on cognitive error theories," in *Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 134-135, Memphis, TN, USA, October 2018.

[15] J. Gao, L. Zhang, F. Zhao, and Y. Zhai, "Research on software defect classification," in *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 748–754, Chengdu, China, March 2019.

[16] Y. Qu, T. Liu, J. Chi et al., "Node2defect: using network embedding to improve software defect prediction," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 844–849, Montpellier, France, September 2018.

[17] S. I. Ayon, "Neural network based software defect prediction using genetic algorithm and particle swarm optimization," in *Proceedings of the 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, pp. 1–4, Dhaka, Bangladesh, May 2019.

[18] E. A. Felix and S. P. Lee, "Predicting the number of defects in a new software version," *PLoS One*, vol. 15, no. 3, pp. 1–30, https://ideas.repec.org/a/plo/pone00/0229131.html.

[19] P. Paramshetti and D. A. Phalke: Survey on software defect prediction using machine learning techniques.

[20] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, 2019.

[21] S.-P. Zhu, B. Keshtegar, and S. Trung, "Novel probabilistic model for searching most probable point in structural reliability analysis," *Computer Methods in Applied Mechanics and Engineering*, vol. 366, p. 113027, 2020.

[22] S.-P. Zhu, B. Keshtegar, M. Bagheri, P. Hao, and N.-T. Trung, "Novel hybrid robust method for uncertain reliability analysis using finite conjugate map," *Computer Methods in Applied Mechanics and Engineering*, vol. 371, p. 113309, 2020.

[23] S.-P. Zhu, B. Keshtegar, K. Tian, and N.-T. Trung, "Optimization of load-carrying hierarchical stiffened shells: comparative survey and applications of six hybrid heuristic models," *Archives of Computational Methods in Engineering*, vol. 28, no. 5, pp. 4153–4166, 2021.

[24] X. Z. Gao, X. M. Gao, and S. J. Ovaska, "A modified Elman neural network model with application to dynamical systems identification," in *Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No.96CH35929)*, vol. 2, no. 14–17, pp. 1376–1381, Beijing, China, October 1996.

[25] J. T.-H. Lo, Y. Gui, and Y. Peng, "The normalized risk-averting error criterion for avoiding nonglobal local minima in training neural networks," *Neurocomputing*, vol. 149, pp. 3–12, 2015.

[26] M. Ltaief, H. Bezine, and A. M. Alimi, "A spiking neural network model with fuzzy learning rate application for complex handwriting movements generation," *Advances in*

Intelligent Systems and Computing, vol. 552, pp. 403–412, 2017.

[27] K. G. Kapanova, I. Dimov, and J. M. Sellier, "A genetic approach to automatic neural network architecture optimization," *Neural Computing & Applications*, vol. 29, no. 5, pp. 1481–1492, 2018.

[28] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, NJ, USA, 2015.

[29] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[30] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933.

[31] D. K. Choubey, P. Kumar, S. Tripathi, and S. Kumar, "Performance evaluation of classification methods with PCA and PSO for diabetes," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 9, no. 1, pp. 1–30, 2020.

[32] R. Malhotra and K. Khan, "A study on software defect prediction using feature extraction techniques," in *Proceedings of the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 1139–1144, Noida, India, June 2020.

[33] A. Ouaarab, B. Ahiod, and X.-S. Yang, "Discrete Cuckoo Search algorithm for the travelling salesman problem," *Neural Computing & Applications*, vol. 24, no. 7-8, pp. 1659–1669, 2014.

[34] L. Wang, S. Yang, and W. Zhao, "Structural damage identification of bridge erecting machine based on improved Cuckoo search algorithm," *Journal of Beijing Jiaotong University*, vol. 37, no. 4, pp. 168–173, 2013.

[35] H. Zheng and Y. Zhou, "Self-adaptive step Cuckoo search algorithm," *Computer Engineering and Applications*, vol. 49, no. 10, pp. 68–71, 2013.

[36] O. Ciftcioglu and E. Turkcan, "Selection of hidden layer nodes in neural networks by statistical tests," in *Proceedings of the EUSIPCO-92. Sixth European Signal Processing Conference (EUSIPCO)-1992*, Brussels, Belgium, August 1992, http://inis.iaea.org/search/search.aspx?orig_q=RN:2402705.

[37] A. M. Reynolds and M. A. Frye, "Free-flight odor tracking in Drosophila is consistent with an optimal intermittent scale-free search," *PLoS One*, vol. 2, no. 4, p. e354, 2007.

[38] I. Pavlyukevich, "Lévy flights, non-local search and simulated annealing," *Journal of Computational Physics*, vol. 226, no. 2, pp. 1830–1844, 2007.

[39] B. Henderson-Sellers, *Object-oriented Metrics: Measures of Complexity*, Prentice-Hall, Upper Saddle River, NJ, USA, 1995.

[40] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[41] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 414–423, Hyderabad, India, May 2014.

[42] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's Delta Calculator: a non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.