Pattern Recognition Letters 160 (2022) 50–57

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Cross-project defect prediction based on G-LSTM model

Ying Xing[a], Xiaomeng Qian[b], Yu Guan[c,*], Bin Yang[c], Yuwei Zhang[d]

[a] *School of Artificial Intelligence, Beijing University of Posts and Telecommunications, No.10, Xitucheng Road, Beijing, 100876, China*
[b] *School of Modern Post, Beijing University of Posts and Telecommunications, No.10, Xitucheng Road, Beijing, 100876, China*
[c] *Du Xiaoman, Science Technology Co., Ltd., No.10, Xibeiwang East Road, Beijing, 100085, China*
[d] *School of Computer Science, Peking University, No.5, Summer Palace Road, Beijing, 100871, China*

ARTICLE INFO

Article history:
Received 27 December 2021
Revised 20 March 2022
Accepted 28 April 2022
Available online 30 April 2022

Edited by: Maria De Marsico.

Keywords:
Computational language processing
Cross-project defect prediction
Long-term and short-term memory neural network
Continuous bag-of-word model
Generative adversarial network

ABSTRACT

Cross-project defect prediction (CPDP) is currently a hot research direction in the field of software reliability. Traditional CPDP methods cannot capture the semantic and contextual information of programs by handcrafted features, which affects the prediction performance. In this paper, we apply technology in the NLP domain to solve it. We first extract token vectors from the abstract syntax tree (AST) of source and target code files, and then convert them into numerical vectors by the word embedding algorithm of continuous bag-of-word model (CBOW) as the input of the proposed deep learning model named Generative Adversarial Long-Short Term Memory Neural Networks (G-LSTM). The model integrates generative adversarial network (GAN) and bidirectional long-short term memory networks (BiLSTM) with attention mechanism to automatically learn semantic and contextual features of programs. Specifically, GAN is used to eliminate the differences in data distribution between source and target projects, and BiLSTM is the feature extraction encoder. We compose five projects of the PROMISE dataset into 20 source-target project pairs and conduct comparison experiments on them. The experimental results demonstrate that our method outperforms some traditional and state-of-the-art CPDP methods in terms of the evaluation metrics of AUC and Acc.

1. Introduction

Modern software written in computational language is becoming more and more powerful, and its scale and complexity are also increasing. Defects in computational language programming poses a huge threat to the quality and reliability of software. In order to ensure software quality, software defect prediction (SDP) is proposed as an effective means of identifying defects, which not only reduces the cost and time of software testing, but also ensures that the testing team can locate defects more easily [1]. However, the prerequisite for SDP implementation is to obtain sufficient historical data, which is difficult to achieve in the early stage of software development. In order to solve this problem, cross-project defect prediction (CPDP) is presented [2], the main concept of which is to build the defect prediction model according to the defect information of mature projects (source projects), and then apply it to new projects (target projects) to predict software modules prone to defects [3].

Previous researches on CPDP mainly focus on the use of handcrafted static features to establish the defect prediction model by machine learning techniques [4–6]. Those handcrafted features mainly represent the macroscopic statistical characteristics of the code, such as the number of lines of code or average method complexity, etc [7]. The source code of the program contains rich structural and semantic information. However, researches using only handcrafted features are difficult to capture the complex information from the source code, especially, the contextual and semantic details within methods. Therefore, it is almost impossible for handcrafted features to accurately identify defects when defects appear inside methods. This limitation is an important factor affecting the prediction accuracy of CPDP. To overcome the limitation, some researchers [8–11] have introduced techniques from the natural language processing (NLP) domain to CPDP researches to learn meaningful contextual and semantic features from source code. Specifically, the core idea is to represent the source code in the form of abstract syntax trees (AST) [12], then treat the nodes on the trees as words in text, afterwards, utilize the models in the NLP domain to conduct pre-processing on words and extract features, and finally, train classifiers to predict. Commonly used NLP techniques include language models for pre-processing and deep learning models for feature extraction. The former can quantitatively

* Corresponding author.
E-mail addresses: xingying@bupt.edu.cn (Y. Xing), qianxiaomeng@bupt.edu.cn (X. Qian), guanyu@bupt.edu.cn (Y. Guan), yangbin01@duxiaoman.com (B. Yang), yuweizhang@pku.edu.cn (Y. Zhang).

https://doi.org/10.1016/j.patrec.2022.04.039
0167-8655/© 2022 Published by Elsevier B.V.

represent text sequences, such as neural network language model (NNLM) [13], word to vector (Word2vec) [14] and FastText [15]. The latter has a wide variety, including convolutional neural network-based (CNN-based) models [16], recurrent neural network-based (RNN-based) models [17], attention mechanism-based models [18], transformer-based models [19], etc. In [8,9], the comparison experiments with traditional machine learning methods can demonstrate the superiority of NLP techniques.

Another factor that affects the performance of CPDP is the difference in data distribution between projects [20]. Previously, researchers assume that the source project and the target project show the same distribution. In fact, since the projects developed by different teams and companies are always different in terms of scale, function and coding standards, the source data of these projects are inevitably different in distribution. In other words, different projects may have different data distributions. In order to solve the problem, some researches [4–6] have attempted to exploit machine learning methods such as transfer learning. Whereas, those traditional machine learning methods are often unable to learn complex features, and the design of the loss function is complicated. It has been experimentally demonstrated that generative adversarial network (GAN) [21] can be well applied as a tool for eliminating inter-domain differences in areas such as NLP and image recognition [22–24]. Compared with traditional transfer learning methods, GAN has the following advantages: 1) GAN uses two adversarial neural networks as the training criteria, which can back-propagate, and does not rely on inefficient Markov chain methods or approximate inference. Since there is no complex variational lower bound, the training difficulty is greatly reduced and the training efficiency is improved accordingly. 2) GAN utilizes adversarial training to produce clearer and more realistic samples, and exploits discriminators to achieve data transfer, which can avoid the difficulty of loss function design in traditional transfer learning.

In this paper, we propose a model called Generative Adversarial Long-Short Term Memory Neural Network (G-LSTM), whose core idea is to use the adversarial gaming property of GAN network to transfer the features of target projects, and to utilize the bidirectional long-short term memory network (BiLSTM) [25] as the deep learning feature extractor. The relationship between original GAN network and G-LSTM is shown in the Fig. 1. The original GAN generates realistic fake samples through the generator, makes authenticity judgments in the discriminator. It continuously trains the generator until the distribution of real and fake samples is basically the same. The G-LSTM, on the other hand, borrows the framework of the original GAN, and solves the problem of data distribution differences by continuously training the target BiLSTM (i.e., the feature extractor of the target project, which corresponds to the generator of the original GAN) to make the target features have essentially the same distribution as the source features.

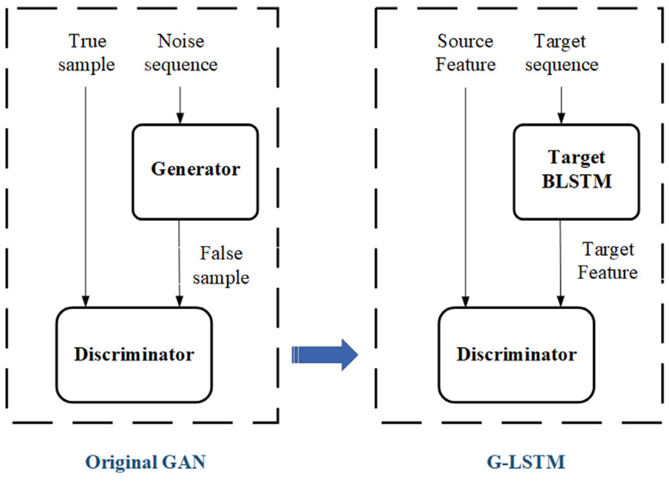


Fig. 1. The relationship between original GAN and G-LSTM.

Specifically, the main contributions of this paper are as follows:

1) Pre-processing techniques from the NLP domain are applied to the CPDP domain. In the data preprocessing stage, we use the continuous bag-of-words (CBOW) [26] model to convert the ASTs of the code into numerical vectors.
2) Combining the GAN structure with the BiLSTM feature extractor to form a new model that enables feature transfer between two projects, thus eliminating inter-domain distribution differences.
3) Experiments are conducted on 20 pairs of source-target projects to evaluate the performance of the proposed model.

2. Related work

We select three most relevant researches to discuss, including application of NLP techniques, development of CPDP, and adversarial learning.

2.1. Application of NLP techniques

Since the handcrafted features lack semantic and contextual information of the source code, in recent years, some researchers have tried to utilize techniques in NLP domain to extract semantic and contextual features in SDP. Yang et al. [8] used Deep Belief Network (DBN) [27] to extract features from the token vectors traversed from the AST of the program source code to build the SDP model, and experimentally demonstrated that the method outperforms the traditional approaches. Wang et al. [9] further mapped each token with a unique integer identifier, transformed token vectors into numerical vectors, and then input them into DBN to extract semantic features. Li et al. [28] proposed an SDP framework based on Convolutional Neural Networks (CNN) [29]. In this framework, they encoded the token vectors into numerical vectors through the word embedding algorithm, and then utilized CNN to automatically learn semantic features. Huang et al. [30] took program semantics as the point of penetration to construct an SDP model based on the attention mechanism, in which a mask model for the correlation between functional methods in program files was introduced. In this paper, we only consider the application of NLP techniques in CPDP scenarios.

2.2. Development of CPDP

To solve the data distribution difference problem, researchers have turned to explore the machine learning methods, and most researches focus on supervised CPDP models using handcrafted features through transfer learning. Pan et al. [4] used transfer component analysis (TCA) to transfer the training data while preserving the data attributes, so that the source data and the target data have similar data distributions. Turhan et al. [5] presented nearest neighbor filtering (NNFilter) to construct a training set identical to the target set for training by pooling together instances with similar feature selection. Zhou et al. [31] proposed a balanced distribution adaptation (BDA) method that considers both marginal distribution and conditional distribution, and adaptively assigns different weights, which is based on transfer learning to balance the data distribution. Tong et al. [32] presented a novel kernel spectral embedding transfer ensemble (KSETE) method, which could find a series of potential common kernel feature subspace. Each of subspace could maximize the similarity between the source and target datasets while preserving the intrinsic characteristics of the data.

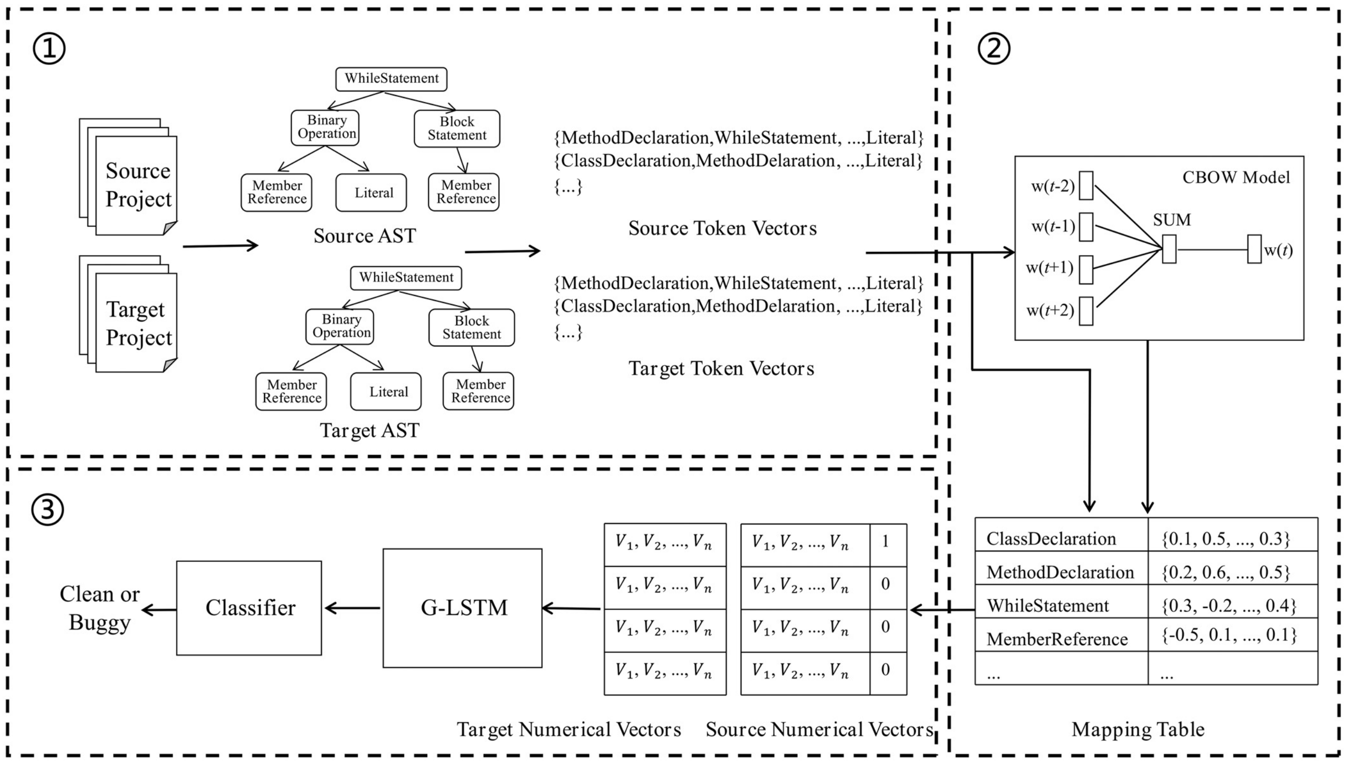


Fig. 2. The framework of the G-LSTM model.

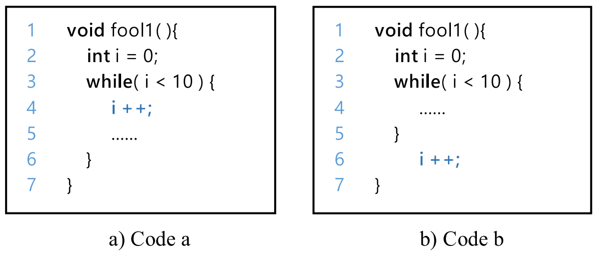
These researches have addressed the data distribution differences problem to varying degrees. However, all those methods are implemented directly on the basis of handcrafted feature extraction, and the design of the loss function is complicated. The method for solving the difference problem applicable to features extracted by neural networks is the focus of this paper.

2.3. Adversarial learning

Adversarial adaptation technology has recently been used as a general tool to minimize the distribution differences between domains. In other fields, there have been researches using generative adversarial network (GAN) to design different generative adversarial algorithms to eliminate inter-domain differences. Bousmalis et al. [22] applied adversarial learning to domain adaptation in natural language process (NLP). They aimed to transfer knowledge from source domain to target domain. Liu et al. [23] proposed coupled generative adversarial networks (CoGANs) that train two GANs to generate images from two domains separately. This method could obtain the domain invariant feature space of both domains by binding the high-level parameters of two GANs. Zeng et al. [24] attempted to establish a unified framework based on the unsupervised domain adaptation technology of adversarial learning objectives, which is called adversarial discriminatory domain adaptation (ADDA), and achieved good experimental results in the experiments across four domain shifts. In this paper, we plans to use the neural network model built by GAN to solve the problem of data distribution differences between source and target projects in CPDP research.

3. Methodology

In this section, we mainly discuss how to implement the G-LSTM model. Fig. 2 illustrates the overall procedure of our proposed method that automatically learns the semantic and contextual features from the source code. Specifically, the G-LSTM model contains several steps: ①Program code parsing; ② AST node mapping and embedding; ③Transfer model building and defect prediction.

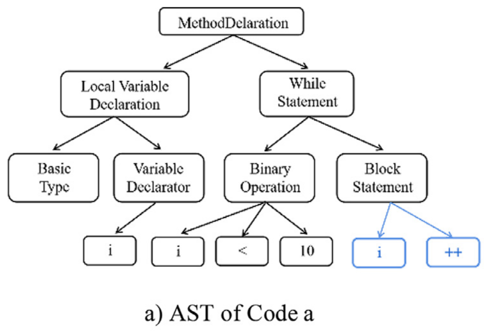

```
void fool1( ){
  int i = 0;
  while( i < 10 ) {
     i ++;
     ......
  }
}
```
a) Code a

```
void fool1( ){
  int i = 0;
  while( i < 10 ) {
     ......
  }
     i ++;
}
```
b) Code b

Fig. 3. Two code examples.

3.1. Program Code Parsing

The first step of the G-LSTM model is to parse the code of each module of the source and target projects into the form of AST. AST is a tree-like representation of the abstract syntax structure of the code, where each statement in the code is represented as a node in the tree. AST can effectively preserve the syntactic structure and semantic information of the code. As shown in Fig. 3, the structure of code *a* and code *b* are very similar. The length of the code, the number of operands and operators, and even their complexity remain the same, so that they have almost the same static code metrics (handcrafted features). That means that there is almost no difference in their statistical features, and the distances between the feature spaces will be very small. In the classification phase, the classifier is likely to group the two codes together. However, it is clear that code *a* is free of defects, while code *b* is defective, which indicates that using only handcrafted features alone will result in wrong detection results. However, the contextual information in these two code fragments is significantly different. Specifi-



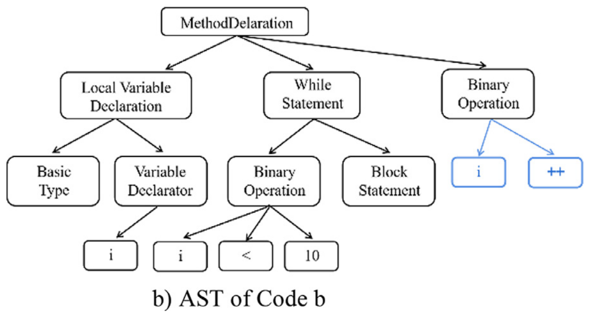


Fig. 4. ASTs of two code examples.

cally, in code *a*, the statement *i*++ is within the *while* loop structure, while in code *b*, it follows the *while* loop structure. Such difference is evident in their ASTs, as shown in Fig. 4. In the AST of code *a*, the node *whileStatement* is the parent of the node for statement *i*++, while in the AST of code *b* they are juxtaposed. Hence, it is necessary to extract semantic and contextual features from the AST of the source code.

In this paper, we exploit Javalang [33], an open source Python dependency package, to parse the code of the source and target projects. Javalang provides a tokenizer and a parser based on the Java language specification, which can help to construct the AST of Java code.

3.2. AST node mapping and embedding

After extracting the ASTs of the source and target projects, we need to traverse each node of the tree to generate the token vectors. There are 92 types of nodes in AST, but we only need nodes that are prone to defects. Therefore, we choose the following four classes of types of nodes as our tokens:

1) nodes that represent method invocations;
2) nodes for declarations, including method declarations, class declarations, interface declarations, etc;
3) control flow nodes, such as if statement, while statement, etc;
4) other nodes.

Since we plan to use deep learning to extract semantic features and token vectors cannot be directly input to the neural network, we need to convert token vectors into numerical vectors. Firstly, we have to construct a dictionary based on the extracted tokens, then utilize word embedding techniques to represent each token as a high-dimensional vector that contains contextual information about each token, and finally exploit the neural network to learn the relationships between the nodes. In this process we use the CBOW model for word embedding. It is a neural network model widely used in the field of NLP for fast training to obtain word vectors, and the core principle is to predict the central word by the first *R* words and the last *R* words of the central word. The structure of CBOW model is shown in Fig. 5.

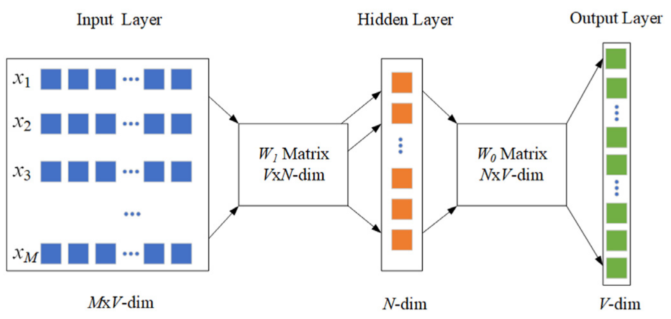


Fig. 5. Structure of CBOW model.

Here the input layer is composed of the context vectors $\{x_1, x_2, ..., x_M\}$ encoded by one-hot, where the window size is M and the vocabulary size is V. It is connected to the hidden layer by a $V \times N$-dimensional weight matrix W_I. The hidden layer performs a weighted average of the input vectors with the following equation.

$$h = \frac{1}{M} W_1 \cdot (\sum_{i=1}^{M} x_i) \tag{1}$$

And the hidden layer is connected to the output layer by a $N \times V$-dimensional weight matrix W_0. Thus, the input of the j-th node of the output layer is as follows.

$$u_j = v'_j \cdot h \tag{2}$$

Here, v'_j represents the j-th column of the W_0 matrix. The output of the output layer is the result of normalizing the predicted probability vector for each target word. The output of the j-th node of the output layer is shown in Equation 3.

$$y_{c,j} = p(w_{y,j}|w_1, ..., w_M) = \frac{\exp(u_j)}{\sum_{j=1}^{V} \exp(u'j)} \tag{3}$$

Based on the output value, the word vector model can be trained by updating the two weight matrices in the form of back propagation. After the training is completed, the trained weight matrix $W_1{}^*$ is the word embedding matrix, and the word vector of the i-th target word is the i-th row of the word embedding matrix.

3.3. Transfer model building and defect prediction

The original GAN network is composed of two networks: a generative network and a discriminative network. In the generative network, the generator G generates a fake data distribution based on a sequence of random noise. Then the generated fake data and the real data are fed together into the discriminative network. Afterwards, the discriminator D in the discriminative network is responsible for determining whether the input is fake or true. The GAN is formulated as follows.

$$V(D, G) = E_{x \sim P_{data}}[logD(x)] + E_{x \sim P_g}[log(1 - D(x))] \tag{4}$$

Here, x denotes the real samples, P_{data} denotes the distribution of the real samples, P_g denotes the distribution of the samples generated by the generator G, $D(x)$ represents the probability that the discriminator determines that the real data is true. From Eq. 4, it can be seen that when the generator G is fixed, *max V*(*G*,*D*) represents the difference between P_{data} and P_g. In order to find a closer distribution, the optimal G needs to be selected so that this maximum value is minimized, that is, the difference between the two distributions is minimized, as shown in Eq. 5.

$$G^* = arg \min_G \max_D V(G, D) \tag{5}$$

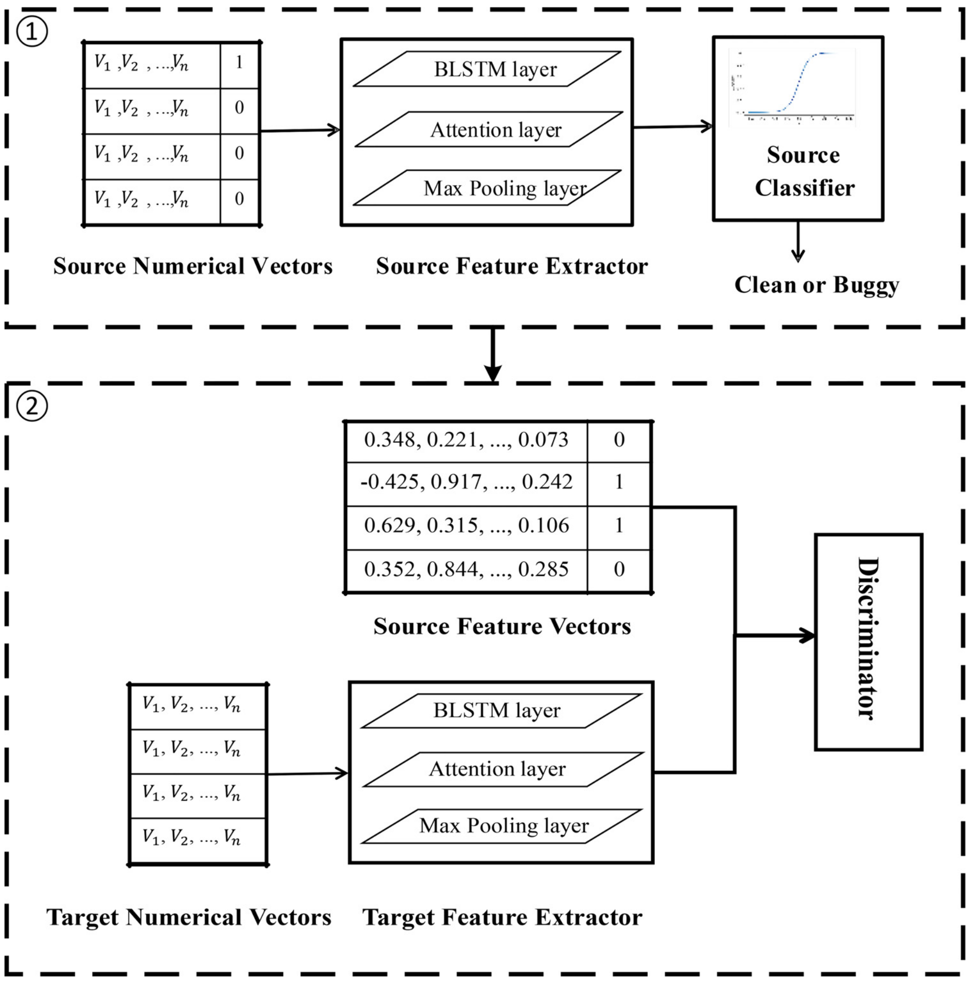


Fig. 6. The framework of transfer model.

Goodfellow et al. [34] demonstrate that GAN has global optimality and optimal performance of the generator G when $P_{data} = P_g$. At this point the generator generates the most realistic data with maximum probability, making the discriminator D unable to distinguish the distribution of training and testing samples. In this paper, the architecture of the original GAN is transformed according to the practical requirements of CPDP, and the deep learning-based transfer model is proposed (shown in Fig. 6).

The basic principle is to treat the source feature vectors as real samples, the target features extractor as generator, and the target feature vectors as fake samples, and then input the source and target feature vectors into the discriminator for gaming. When the discriminator can no longer distinguish the source features from the target features, the difference in distribution between the source and target projects is basically eliminated.

As shown in Fig. 6, the construction process of the transfer model can be divided into two steps. In the first step, we need to train the source feature extractor and source classifier after splitting the source project data into training and validation sets in the ratio of 8:2. The classifier used here is the logistic regression (LR) classifier because it is simple to construct and convenient for subsequent comparison experiments. And the second step requires the use of adversarial learning to eliminate the distribution differences between projects. To shorten the training time, the initial version of the target feature extractor should be the same as the trained source feature extractor. After both the source and target feature vectors are input to the discriminator, the distributional similarity of the source and target projects can be expressed by the cross entropy with the following equation.

$$H((x, y), D) = -y \log D(x_i) - (1 - y) \log(1 - D(x)) \tag{6}$$

In Eq. (6), x denotes the source feature sample, y denotes the target feature sample, and D(x) denotes the probability that the discriminator determines that the source feature sample is true. To obtain the maximum similarity H, the target feature extractor and discriminator need to be trained. First, the target feature extractor parameters need to be frozen and the discriminator needs to be trained according to the gradient ascent method to improve its discriminative ability. If the discriminator can distinguish the source and target samples after training, the discriminator parameters are frozen and the target feature extractor parameters are updated according to the gradient descent method. Afterwards, the target feature vectors are extracted again and input to the discriminator for judgment. So on and so forth until the discriminator determines that all the input samples are of the same class, at which time the distribution of the target feature vectors has converged to the distribution of the source feature vectors.

We select BiLSTM with attention mechanism as our source and target feature extractor, whose structure is shown in Fig. 6, including BiLSTM layers, the attention layer and the max pooling layer. For BiLSTM layer, semantic and contextual features are captured from the input numerical vectors and are then output. For the attention layer, the attention mechanism determines the importance of each node in vectors and assigns a weight to each node when constructing the expression for the labeled vector. The attention mechanism is effective for enhancing the impact of key nodes and

Table 1
The statistics of experimental projects.

Project	Version	# of Files	Defect Rate (%)
Ivy	2.0	352	11.4
Poi	3.0	438	64.1
Xerces	1.4	508	76.8
Synapse	1.2	256	33.6
Xalan	2.6	875	53.1

Table 2
Confusion matrix.

	Real positive	Real negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

learning advanced features. For the max pooling layer, this layer is used to pool an unfixed size feature vector into a fixed size feature vector in the multidimensional space. By using this layer, we can extract representative information from the original feature vector.

Since the source and target feature distributions have converged, the target feature vectors can be directly input to the trained source classifier for prediction.

4. Experimental setup

In this section, we describe the settings of our experiments and evaluate the effectiveness of our G-LSTM model, we put forward two research questions:

- RQ1: Can the word embedding model CBOW widely used in
- the NLP domain learn meaningful vector presentation of tokens in CPDP?
- •RQ2: Can G-LSTM outperform the traditional and state-of-the-art CPDP approaches?

4.1. Datasets

The open source repository PROMISE includes multiple defective open source software projects built in various computational languages. Computational languages are similar to natural languages in that they also contain words that humans can understand, and they also have semantics and sequencing. Some of the techniques used in NLP can also be applied to computational languages. Thus, the repository is widely used in both NLP and CPDP research.

In our case, we choose five open source projects based on the JAVA language from the PROMISE repository as the experimental dataset. In order to ensure the generality of the evaluation results, the chosen projects have different sizes and defect rates. Table 1 shows the specific information of the selected five projects. The chosen projects are combined two by two, which can produce 20 sets of source-target project pairs.

4.2. Evaluation metrics

In this paper, area under the ROC curve (AUC) and accuracy (Acc) are chosen to evaluate the performance of the proposed model. These metrics are widely used to evaluate previous work of CPDP [4–6]. In the process of binary classification, the results can be divided into four categories according to the classification results as shown in Table 2.

Based on the confusion matrix, real positive rate (TPR) and false positive rate (FPR) can be calculated as shown in formulas (7) and (8).

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

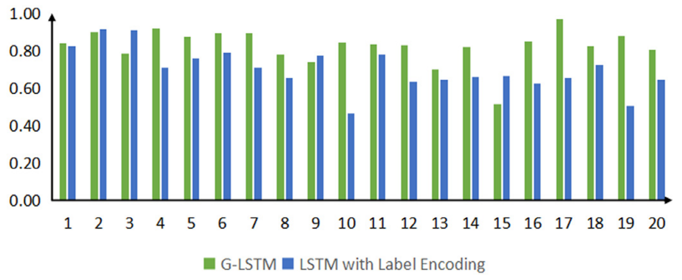


Fig. 7. AUC of the first comparison experiment.

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

Then the ROC curve can be drawn with FPR as the horizontal axis and TPR as the vertical axis. The area under the ROC curve is AUC. According to the definition of AUC, the larger the AUC value, the better the prediction model.

Acc is also positively correlated with model performance, we can define Acc as follows.

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

4.3. Statistical analysis methods

Statistical test can be used to analyze whether there exists a statistically significant difference between results of different methods. We exploit Cliff's delta to measure the effect size between two different methods, which can quantify the amount of difference between them. Table 3 describes corresponding effectiveness levels for different ranges of values of Cliff's delta.

5. Experimental design and results analysis

This section discusses the experimental design for the two RQs and the corresponding analysis of the experimental results. In this paper, we use Python environment and deep learning framework TensorFlow to implement the proposed model. All experiments are run on a Linux server with NVIDIA RTX 2080. The total number of lines of code for the G-LSTM model is 1607. And the G-LSTM model was run for 50 epochs on each of the 20 sets of experiments.

5.1. RQ1: Can the word embedding model CBOW widely used in the NLP domain learn meaningful vector presentation of tokens in CPDP?

In RQ1, we want to investigate whether the word embedding model CBOW can learn meaningful vector presentation for tokens and help to improve the prediction performance of our proposed model G-LSTM. Specifically, we design a comparison experiment between the GLSTM method and the method that uses simple sequential number encoding (also called label encoding) directly for classification. And we evaluate the function of the CBOW model from qualitative perspective and quantitative perspective respectively. The experimental results are shown in Fig. 7 and Table 4.

Table 4
The statistics of the first comparison experiment.

	Average AUC	Cliff's delta
LSTM with Label Encoding	0.702	0.638
G-LSTM	**0.824**	-

Table 5
The statistics of the second comparison experiment.

	Average AUC	Average Acc	Cliff's delta-AUC	Cliff's delta-Acc
Baseline1	0.560	0.584	0.918	0.680
Baseline2	0.633	0.601	0.865	0.630
Baseline3	0.536	0.467	0.925	0.705
G-LSTM	**0.824**	**0.709**	-	-

Table 3
Effectiveness Levels for Different Cliff's Delta.

Value Range	Effectiveness Level
$0.474 \le \lvert\delta\rvert$	Large
$0.33 \le \lvert\delta\rvert < 0.474$	Medium
$0.147 \le \lvert\delta\rvert < 0.33$	Small
$\lvert\delta\rvert < 0.147$	Negligible

From the qualitative perspective, the word embedding model CBOW considers the contextual information of the nodes and facilitates the subsequent processing of extracting semantic and contextual features. Theoretically, the application of CBOW model in CPDP makes sense. From the quantitative perspective, as can be seen from Fig. 7, the AUC values of G-LSTM in 80% of the 20 sets of experiments are better than those of the label encoding-only method. As is shown in Table 4, the average AUC value of G-LSTM is 17.38% higher than that of label encoding method, which reflects the large difference between them. In addition, the value of Cliff's delta is 0.638, which indicates the effect size between the two methods is large.

5.2. RQ2: Can G-LSTM outperform the traditional and state-of-the-art CPDP approaches?

In RQ2, we seek to investigate whether the proposed G-LSTM model can outperform other CPDP baselines. Specifically, we choose several traditional and state-of-the-art CPDP baselines as follows. All comparison experiments were performed in Matlab 2016a environment.

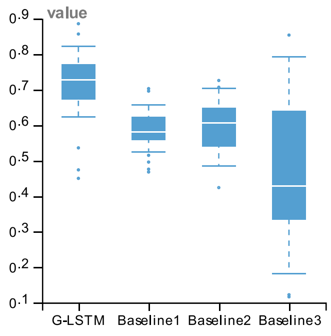
Baseline1 (TCA [4]): The baseline1 method is a feature-based transfer learning method that minimizes the data distance between source and target data to achieve similar distribution of them. (973 lines of code).

Baseline2 (BDA [31]): The baseline2 method adaptively assigns different weights to marginal distribution and conditional distribution, and makes the source and target data distributions similar through data transferring. (280 lines of code).

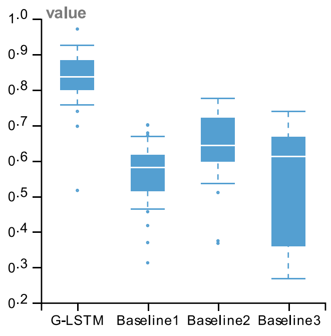
Baseline3 (KSETE [32]): The baseline3 method combines kernel spectrum embedding, transfer learning and ensemble learning to improve model performance. (927 lines of code).

As shown in Fig. 8, the white line indicates the median of 20 sets of the experimental values, and G-LSTM has the highest median and the highest box plot position in two subgraphs, which means that the overall experimental performance of G-LSTM on both AUC and Acc outperformed the other baselines. Besides, as can be seen in Table 5, the computed values of Cliff's delta of baseline1, baseline2 and baseline3 are all greater than 0.474, which reflects that the level of variation among G-LSTM and other baselines are large. And both the average AUC and average Acc of G-LSTM are higher than the other baselines by more than 15%, proving the better performance of G-LSTM.

Theoretically, G-LSTM transfers more adequately due to the adversarial training approach that can produce clearer and more realistic samples. Moreover, G-LSTM considers the extraction of semantic and contextual features, and is more sensitive to defects, thus, its prediction performance is better.



a) AUC



b) Acc

Fig. 8. The box-plot of the second comparison experiment.

6. Conclusion and future work

In this paper, we propose a new deep learning-based CPDP model, namely, the G-LSTM model. The main process of the G-LSTM model is as follows: We first use a simplified AST to represent the code of each extracted program module. The token vectors are then traversed by ASTs and word embedding are performed using the CBOW algorithm, which help token vectors to be converted into numerical vectors. Then the numerical vectors are fed into the G-LSTM model to eliminate cross-project data distribution differences and extract the semantic and contextual features of the target project. Finally, the presence of defects is determined by LR classifier. In order to verify the effectiveness of our model, we adopt AUC and Acc as evaluation metrics for the two sets of comparison experiments. Experimental results show that the proposed G-LSTM model outperforms the chosen traditional and state-of-the-art CPDP methods. The accuracy of the model prediction is somewhat limited because we only use the most basic LR classifier. In the future, we plan to use ensemble learning that has better classification performance. Besides, the G-LSTM model is only applicable to one-to-one prediction, and we can try to study the many-to-one case in the future.

Declaration of Competing Interest

None.

Acknowledgments

The authors thank the anonymous reviewers for their feedback which helped improve this paper. This work has been partially supported by the National Natural Science Foundation of China (No. 61702044) and the National Key Research and Development Program of China (No.2017YFD0401001).

References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, in: A Systematic Literature Review on Fault Prediction Performance in Software Engineering, 38, IEEE Transactions on Software Engineering, 2012, pp. 1276–1304.

[2] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, Software defect prediction using static code metrics underestimates defect-proneness, in: the 2010 International Joint Conference on Neural Networks (IJCNN), 2010, pp. 1–7.
[3] S. Hosseini, B. Turhan, D. Gunarathna, in: A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction, 45, IEEE Transactions on Software Engineering, 2019, pp. 111–147.
[4] J Nam, S J Pan, S Kim, Transfer defect learning, in: 2013 35th international conference on software engineering (ICSE), IEEE, 2013, pp. 382–391.
[5] B Turhan, T Menzies, A B Bener, et al., On the relative value of cross-company and within-company data for defect prediction, Empir. Softw. Eng. 14 (5) (2009) 540–578.
[6] Y. Zhang, D. Jin, Y. Xing, Y. Gong, Automated defect identification via path analysis-based features with transfer learning, J. Syst. Software 166 (2020) 110585.
[7] X. Xia, D. Lo, S.J. Pan, N. Nagappan, X. Wang, Hydra Massively compositional model for cross-project defect prediction, IEEE Trans. Softw. Eng. 42 (2016) 977–998.
[8] X. Yang, D. Lo, X. Xia, Y. Zhang, J. Sun, Deep Learning for Just-in-Time Defect Prediction, in: 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015, pp. 17–26.
[9] S. Wang, T. Liu, L. Tan, Automatically learning semantic features for defect prediction, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 297–308.
[10] M. Massoudi, N.K. Jain, P. Bansal, Software defect prediction using dimensionality reduction and deep learning, in: 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 884–893.
[11] K Shi, Y Lu, G Liu, et al., MPT-embedding: An unsupervised representation learning of code for software defect prediction, J. Software 33 (4) (2021) 23–30.
[12] A Hindle, E.T Barr, M Gabel, Z Su, P Devanbu, On the naturalness of software, Commun. ACM 59 (5) (2016) 122–131.
[13] Y Bengio, R Ducharme, P Vincent, et al., A neural probabilistic language model, J. Mach. Learn. Res. 3 (2003) 1137–1155.
[14] T Mikolov, K Chen, G Corrado, et al., Efficient estimation of word representations in vector space, 2013 arXiv:1301.3781.
[15] P Bojanowski, E Grave, A Joulin, et al., Enriching word vectors with subword information, Trans. Assoc. Comput. Linguist. 5 (2017) 135–146.
[16] A Conneau, H Schwenk, L Barrault, et al., Very deep convolutional networks for text classification, in: In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, 1, Long Papers, 2017, p. 1107−1116.
[17] Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations. arXiv:1802.05365.
[18] H Y Choi, K Cho, Y H Bengio, Fine-grained attention mechanism for neural machine translation, Neurocomputing 284 (2018) 171–176.
[19] T Wolf, L Debut, V Sanh, et al., Transformers: state-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, Demos. Stroudsburg: ACL, 2020, pp. 38–45.
[20] J Nam, S.J Pan, S Kim, Transfer defect learning, in: 2013 35th international conference on software engineering (ICSE), IEEE, 2013, pp. 382–391.
[21] K Wang, C Gou, Y Duan, et al., Generative adversarial networks: introduction and outlook, IEEE J. Automatica Sinica 4 (4) (2017) 588–598.
[22] K Bousmalis, G Trigeorgis, N Silberman, D Krishnan, D Erhan, Domain separation networks, 2016 arXiv preprint arXiv:1608.06019.
[23] M Y Liu, O Tuzel, Coupled generative adversarial networks, 2016 arXiv preprint arXiv:1606.07536.
[24] E Tzeng, J Hoffman, K Saenko, T Darrell, Adversarial discriminative domain adaptation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7167–7176.
[25] A Ray, S Rajeswar, S Chaudhury, Text recognition using deep BLSTM networks, in: 2015 eighth international conference on advances in pattern recognition (ICAPR), IEEE, 2015, pp. 1–6.
[26] T Mikolov, K Chen, G Corrado, J Dean, Efficient estimation of word representations in vector space, 2013 arXiv preprint arXiv:1301.3781.
[27] G E Hinton, S Osindero, Y W Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[28] J. Li, P. He, J. Zhu, M.R. Lyu, Software defect prediction via convolutional neural network, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 318–328.
[29] I Goodfellow, Y Bengio, A Courville, Deep learning, Cambridge: MIT press, 2016.
[30] J. Huang, X. Guan, S. Li, Software defect prediction model based on attention mechanism, in: 2021 International Conference on Computer Engineering and Application (ICCEA), 2021, pp. 338–345.
[31] Z Xu, S Pang, T Zhang, et al., Cross project defect prediction via balanced distribution adaptation-based transfer learning, J. Comput. Sci. Technol. 34 (5) (2019) 1039–1062.
[32] H. Tong, B. Liu, S. Wang, Kernel spectral embedding transfer ensemble for heterogeneous defect prediction, IEEE Trans. Software Eng. 47 (9) (2021) 1886–1906.
[33] Javalang, 2020. https://github.com/c2nes/javalang.
[34] I.J Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, et al., Generative adversarial networks, 2014 arXiv:1406.2661.