Hindawi
Scientific Programming
Volume 2021, Article ID 8381550, 7 pages
https://doi.org/10.1155/2021/8381550

Research Article

Malware Detection Using CNN via Word Embedding in Cloud Computing Infrastructure

Rong Wang, Cong Tian, and Lin Yan

School of Computer Science and Technology, Xidian University, Xi'an 710000, China

Correspondence should be addressed to Rong Wang; bilywr@163.com

Received 5 August 2021; Accepted 2 September 2021; Published 13 September 2021

Academic Editor: Punit Gupta

The Internet of Things (IoT), cloud, and fog computing paradigms provide a powerful large-scale computing infrastructure for a variety of data and computation-intensive applications. These cutting-edge computing infrastructures, however, are nevertheless vulnerable to serious security and privacy risks. One of the most important countermeasures against cybersecurity threats is intrusion detection and prevention systems, which monitor devices, networks, and systems for malicious activity and policy violations. The detection and prevention systems range from antivirus software to hierarchical systems that monitor the traffic of whole backbone networks. At the moment, the primary defensive solutions are based on malware feature extraction. Most known feature extraction algorithms use byte N-gram patterns or binary strings to represent log files or other static information. The information taken from program files is expressed using word embedding (GloVe) and a new feature extraction method proposed in this article. As a result, the relevant vector space model (VSM) will incorporate more information about unknown programs. We utilize convolutional neural network (CNN) to analyze the feature maps represented by word embedding and apply Softmax to fit the probability of a malicious program. Eventually, we consider a program to be malicious if the probability is greater than 0.5; otherwise, it is a benign program. Experimental result shows that our approach achieves a level of accuracy higher than 98%.

1. Introduction

Cloud-fog-edge computing, especially cloud computing, is providing a variety of services in many areas throughout the world. The cloud offers a variety of unique security problems, one of which is the detection of malware. Malware is a significant threat to modern computing devices for their illegal purposes, such as unauthorized access, stealing confidential or personal information, implanting ads, and disrupting normal operation. Therefore, some effective approaches and tools for detecting and deactivating malware are required.

A widely used approach for malware detection is behavior-based method, which monitors the behaviors of a program, typically the stream of system calls, to determine whether it is malicious [1, 2]. However, behavior-based malware detection method is inferior in detecting the unseen and continuously modified malware because of its rigid and restrictive nature [3].

In this article, we propose a framework for malware detection that combines behavior-based feature extraction with deep learning. Our work makes the following major contributions:

(1) Unified design of behavior detection and feature extraction: we use a tool Cuckoo Sandbox to collect the behavior information of the program (executable file) when it executed inside a realistic but isolated environment. Then, we construct a map of monitored behavior to a vector space with a method GloVe [4], which trains on a global word-word co-occurrence matrix and produces a word vector space model, to represent each word of the corpus with a real-valued vector.

(2) Representation of the program's behavioral characteristics as a feature map (a matrix consisting of sequential word vectors): this transformation makes behavioral patterns reflected geometrically and

accessible by convolutional neural network, an excellent deep learning algorithm to capture word-level features from the feature maps and further predict the labels of the feature maps, malware or benign.

The rest of this article is organized as follows. Section 2 describes the related work of malware detection. Section 3 studies the problem formulation posed by our approaches. Section 4 introduces the framework of the proposed malware detection model in detail. Section 5 presents the experimental results that are followed by the conclusions in Section 6.

2. Related Work

Malware detection has been a critical challenge in computing since the late 80s, which mainly involves two processes, feature extraction and classification. For feature extraction, many researches focus on using information available in API calls to monitor the behavior of the program that may potentially highlight anomalous and malicious activities.

The work by Alazab et al. [5] studied an automated method of extracting API call features and analysed them to understand their use for malicious purpose. Sundarkumar et al. [6] presented a model, based on the types of API call sequences, using text mining and topic modeling to detect malware. Hachinyan [7] discussed proactive methods based on API call sequences analysis and proposed a method using a multiple sequence alignment to identify malware. Most recently, Pektas, and Acarman [8] presented a runtime behavior-based classification approach for Windows malware, which extracts runtime behaviors for the determination of a malicious sequence of API calls. In this article, we extract API call sequences of program as a baseline to compare with extracting feature from the behavioral information of program in a sandbox environment.

Term Frequency-Inverse Document Frequency (TF-IDF) is a common weighted technique for information retrieval and data mining. Term Frequency (TF) refers to the number of times a given word appears in a document. Jones [9] first puts forward a technical term, later known as Inverse Document Frequency (IDF), which counts the documents containing (or being indexed by) the term in question. TF-IDF has been used for web document clustering and ranking [10], text classification [11], analysis of similarity between important terms in text documents [12], and image retrieval [13]. In this article, we use TF-IDF to extract feature from the corpus of program behavior descriptions as another baseline compared with word embedding and use SVM to classify the numerical characteristics compared with CNN classifying feature maps.

3. Problem Formulation

Several malware detection researches related to Deep Learning have been proved effective, which extract information from log files of program, like API call sequences during execution process, and create a so-called program behavior language model based on this information. But there are two problems:

(1) It is not certain that the resulting textual information from program's log files can describe the nature of programs

(2) The behavior language model established using the GloVe model for API call sequences may lose the syntax information in some dimension

As we all know, the quality of feature extraction has a decisive influence on malware detection. For accuracy and effectiveness, we want more comprehensive, expressive, and available features of programs. On the other hand, byte data in malware can contain multiple types of information, including human-readable text, binary code, images, and even some encrypted content. Therefore, we expect to design a malware detection model that can extract sufficient information from program and be described in a usable format.

4. Methodology

In this section, we propose a new malware detection method with supervised learning. Figure 1 shows an overview of the main steps in our method. This model adopts 5-fold cross validation to process datasets (malware program set and benign program set).

In the training phrase above the dotted line, we first adopt Cuckoo sandbox to extract behavior information of the executable files, such as "Installs itself for autorun at Windows startup," "Installs itself for autorun at Windows startup." Then, we perform a preprocessing to get word sequences, which called Information Unit. We use Glove model to capture the rich semantic and syntactic features of words and vectorize them. The output of GloVe is a dictionary of word embedding for each unique word. After that, we look up in the dictionary to represent every word in the description file as its corresponding vector, thus a feature map of the program is generated. Finally, we obtain a trained CNN by learning from these feature maps.

Below the dotted line, the flowchart describes the detection phrase of our model. More specifically, we first input the executable file from the test set into Cuckoo for analysis to get the Information Unit and then look up in the dictionary of word embedding to get the Information Vector Unit; finally, we use CNN model mentioned above to predict the probability that the executable file is benign or malware. If the probability is higher than 0.5, we consider the program as a benign program, otherwise a malicious program.

4.1. Word Embedding. This stage first utilizes Cuckoo to analyze the executable files that may contain some malicious codes and obtains the program behavior information that described the natural language. Then, the model use GloVe proposed by Jeffrey Pennington to complete the word embedding. Each word w is represented by a 100-dimensional vector v. During this step, we leverage global statistical information by training on the word-word co-occurrence matrix X, where the element X_{ij} presents the number of

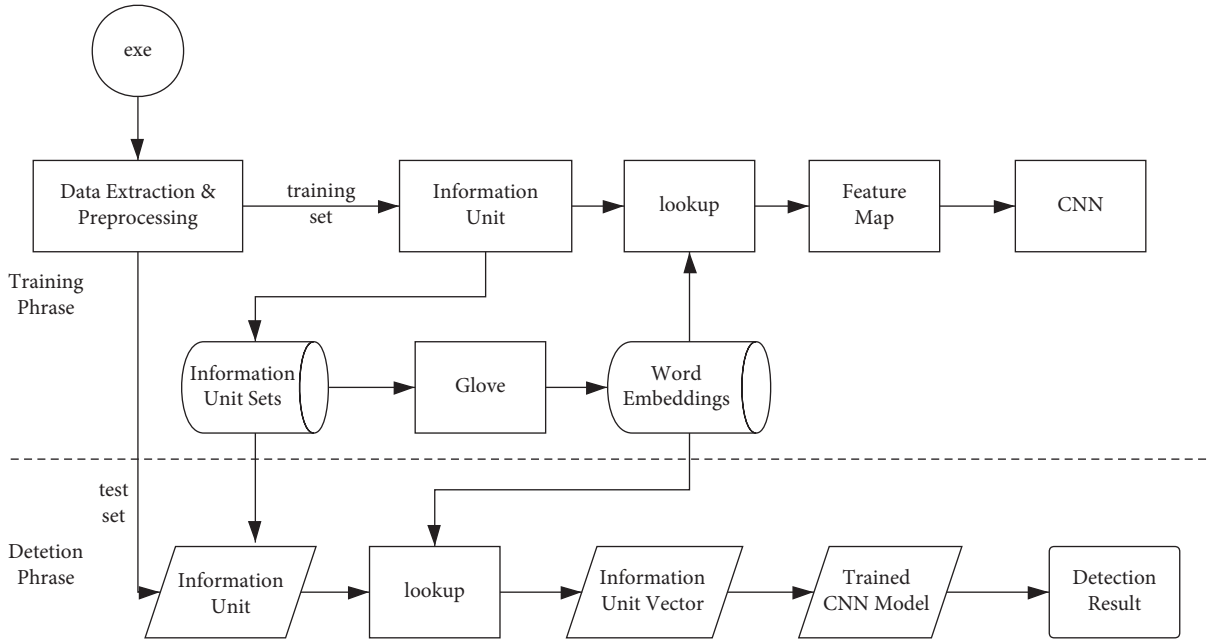

Figure 1: Overview of the main steps.

times that word w_i occurs in the context of word w_i. Let $X_i = \sum_k X_{ik}$ be the number of times that any word appears in the context of word w_i, and

$$P_{i,j} = P\left(w_j \mid w_i\right) = \frac{X_{ij}}{X_i}, \tag{1}$$

be the probability that word w_j appears in the context of word w_i. For three words w_i, w_j, and w_k, the radio P_{ik}/P_{jk} depends on the co-occurrence frequency of words (w_k, w_i) and (w_k, w_j), and the value is contributed to distinguishing relevant words from irrelevant words or discriminate the two relevant words. Let v_i, v_j, and v_k be the word vectors of w_i, w_j, and w_k, since the inherently linear structures of vector space, the radio P_{ik}/P_{jk} can be considered as a function of v_i, v_j, and v_k. Then, the radio P_{ik}/P_{jk} can be measured as follows:

$$F\left(\left(v_i - v_j\right)^T v_k\right) = \frac{X_{ik}}{X_{jk}}. \tag{2}$$

The regression connection function of the model is $P_{ik} = \exp\left(v_k^T v_i\right) + a_i + a_k$, where a_i and a_k can be approximated as $\log\left(X_i\right)$ and $\log\left(X_j\right)$. Then, $v_k^T = \log\left(P_{ik}\right) - \log\left(X_i\right) - \log\left(X_k\right)$ comes naturally. Then, the cost function of Model is $J = \sum_{i,j=1}^{N} f\left(X_{i,j}\right)\left(v_i^T v_j + a_i + a_j - \log\left(X_{i,j}\right)\right)^2$, where N is the size of the vocabulary (the dimensions of co-occurrence matrix is $N * N$, and v_i, v_j are the vector representations of the words w_i, w_j.

In addition, a weighting function $f(x)$ is needed so that the frequent co-occurrence word pairs will not be overweighted. Various functions can be selected, but we found one work well, which can be parameterized as follows:

$$f(x) = \begin{cases} \left(\frac{x}{x^{\max}}\right)^{\alpha}, & \text{if } x < x_{\max}, \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

Here, $\alpha = 3/4$, and then, the model will perform better to represent the words as vectors containing as many semantic and grammatical information as possible. After getting this word embedding, we can look up in the dictionary to turn word sequences into a two-dimensional feature map, which can also be generated by connecting as:

$$G = v_i \oplus v_j \oplus \cdots \oplus v_k \oplus v_l. \tag{4}$$

The resulting feature map is used as input of CNN.

Figure 2 shows two sample benign feature maps and two sample malware feature maps by their digital gray images. Here, the dimension of a word vector is 100, and different text lengths lead to different image sizes.

4.2. The CNN Architecture. In this work, we treat the feature images as input and perform a convolutional neural network to classify the images. Figure 3 shows the architecture of our CNN used in our model. In general, the basic structure of CNN consists of two layers: feature extraction layer and feature mapping layer. In the feature extraction layer, each neuron's input is connected to the local receptive field of the previous layer, and the local characteristics are extracted. Once the local feature is extracted, the relationship with other characteristics is also determined; In the feature mapping layer, each computing layer of the network consists of multiple feature maps, which can be seen as a plane. The feature mapping structure in our model uses the nonlinear activation function Relu:

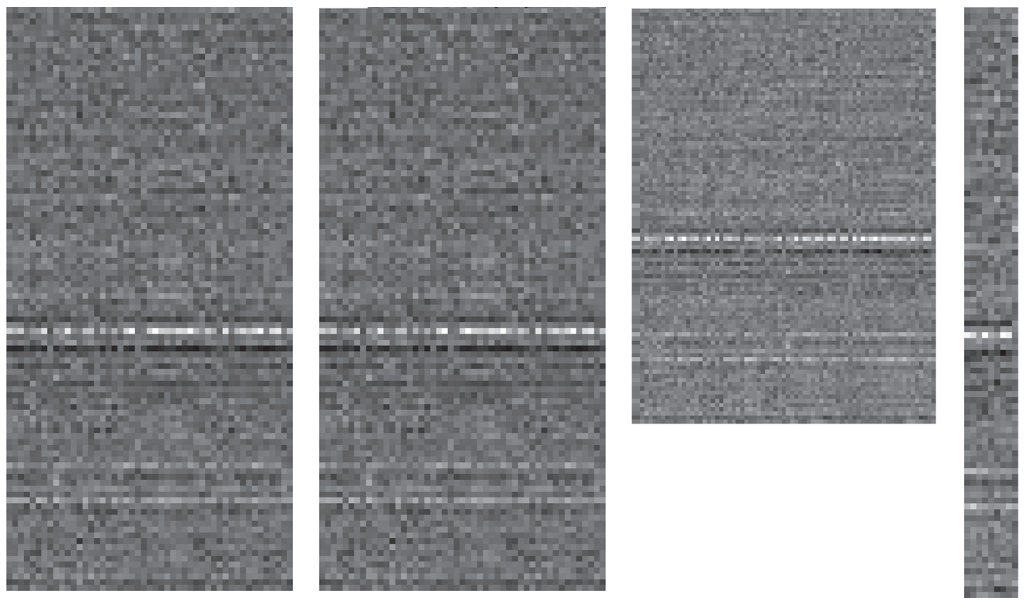

Figure 2: Gray images of two benign feature maps and two malware feature maps.

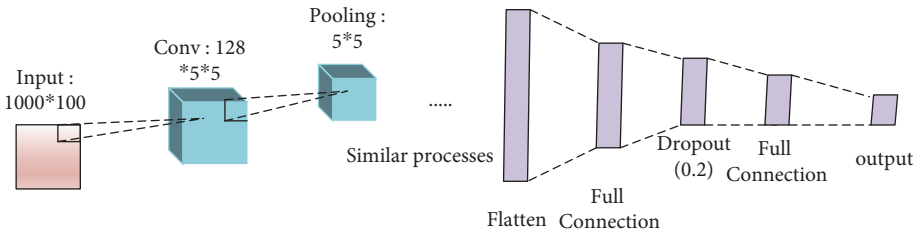

Figure 3: The framework of CNN.

$$f(x) = \max\{0, x\}. \tag{5}$$

4.2.1. Features Extraction. The main task of this stage is to generate a feature map from the complex features within the image. This features extraction stage can be defined as an alternate chain of two different layers (the convolutional layer and the pooling layer).

The convolutional layer applies a series of trainable and learnable K filters (or kernels) to analyze the image. In our CNN, we use the convolutional layer with 128 filters, select the window size of neural network is 5×5, and choose the Relu as the activation function. Each convolution layer in CNN is followed by a calculation layer aimed at the local average and secondary extraction. In the forward calculation, we input a certain size of data (width $w = 1000 *$ height $h = 100$), which dot product with the filter $w \in R^{h \times k}$ (a vector of $h \times k$ dimensions) and add a bias $b \in R^h$.

Then, the pooling layer is designed to compress the data hat produced by the previous convolutional layer and maintain the most relevant features. This layer swipes the filters one by one to form a new output data. More generally, the input is defined by the following parameters: h, w. Here, h represents the height of the volume and w is the width.

When we consider the image processing, for each filter k, the convolutional layer applies a convolution defined as follows:

$$o_{i,j,k} = \sum_{h=0}^{h_k} \sum_{m=0}^{w_k} \left(w_{h,m} \cdot x_{i+h,j+m} \right) + b_k, \tag{6}$$

where the filter k is represented by a $h_k \times w_k$ matrix of weights, b_k is a bias, i and j are the coordinates of the current pixel x in the input volume.

After each convolution layer, we use the pooling method to shrink the parameter space of CNN and filter noises. In this article, we choose the Max-Pooling to obtain the most representative local features. The max-pooling is defined to extract the most critical feature values of the input vectors within a window. So the maximum value of each feature map is obtained as the local optimal feature. After the final flattening, we will get a one-dimensional vector as input for next stage.

4.2.2. Classification. The classification stage receives the input vector from the last layer (convolutional or pooling) of the features extraction stage and then calculates the affinity of the feature map with the classification classes. This stage is

structured as a chain of linear layers, which implemented by a Fully Connected Network (FCN). In our model, there are two Fully Connected Networks and one dropout layer to prevent overfitting.

Suppose that the linear layer is composed by J neurons, which are responsible for aggregating the information derived from the previous layer. Then, the output values are expressed as a weighted linear combination of these neurons:

$$o_j = \sum_{i=0}^{I} \left(w_{i,j} \cdot x_i \right) + b_k. \tag{7}$$

Here, $w_{i,j}$ represents the weight, x_i represents the neurons from the previous layer, and b_j represents a bias.

As for the dropout layer, we found that the model works well when the dropout rate was equal to 0.2. The last normalization operator receives the output from the last linear layer and calculates the affinity of the feature image with the classification classes in percentage terms using a SoftMax operator σ: $\sigma_j = e^{x_j} / \sum_{k=1}^{K} e^{x_k}$ for $j = 1, \ldots, K$. Here, x_j means the output of the last linear layer. Then, SoftMax operator enforces the output in range [0, 1]. Thus, this normalized operator output can be seen as the probability of classification.

4.3. Network Training. Convolutional neural network is a supervised learning algorithm. Due to our target of malware detection, the training data should consist of sufficient examples of two classes of application program, malicious program and benign program, so that CNN can learn to capture the relevant features of malware program for further classification more effectively. Fortunately, we have found an efficient method of feature extraction, which collects the global and local statistic information of executable programs. Therefore, the next critical step is to train the vectored data (or feature map) using CNN.

4.4. Dataset and Parameter Settings. In this work, we collected malware samples from some antimalware communities (e.g., Kafan Forum and MalShare) and obtained some benign system programs built in Windows, benign applications from the download site of software (e.g., Greenxf and PConline). Our dataset consists of 1992 programs (executable files), whose size ranges from a few KB to a dozen megabytes. Among them, there are 981 malicious programs, which are considered as negative examples, and 1011 benign programs, which are treated as positive examples. For these programs, we choose 4/5 of dataset as the training set and 1/5 as the testing set, to evaluate the classification results of our model. Batchsize and epoch we choose are 128 and 4, separately. In training process, we use the convolutional layer with 128 filters and the pooling layer alternately before the flatten, after that there are two Full Connection layers and a Dropout behind them for avoiding overfitting and improving the generalization ability of the network.

5. Experiment

We conduct a series of experiments to evaluate the efficiency of our approach. As mentioned before, we extract the textual information by Cuckoo from the program files and describe the behavior of programs in natural language, for example, "Allocatesread-write-execute memory (usually to unpack itself)". It is very critical for us to map the words into the vector space and represents each word with a 100-dimensional vector, which will produce a feature map for each program. Then, we can develop CNN to detect malware with the feature maps obtained before as input. That is the whole process of our model, we will denote it as Method B in the below. In order to compare the efficiency of our model, we design two other groups of experiments: Method A and Method C. Method A also extracts textual information describing the behavior of programs by Cuckoo in natural language but generates the feature vectors by TF-IDF as the input of SVM for malware detection. However, Method C extracts API call sequences of programs and uses GloVe to get the feature map and then uses CNN to detect malware.

5.1. Performance Comparison. As our approach relies on machine learning techniques, we also follow the model evaluation method to assess our experimental results. In this work, our mainly used measurement parameters are Accuracy, Precision, Recall, F1, Kappa, and ROC Curve, for a more comprehensive assessment. For binary classification task, we can divide the sample set into four classes: true positive (TP), false positive (FP), true negative (TN), and false negative (FN), according to the combination of real class and predicted class.

Accuracy is the proportion of the correct sample in the total sample. For a sample set D, the accuracy is defined as accuracy = TP + TN/TP + FN + TN + FP. Precision is defined as P = TP/TP + FP. Recall is defined as R = TP/TP + FN. Other performance measures take the precision and the recall into account at the same time, such as F1 and kappa. F1 is defined as $F1 = 2 * \text{precision} * \text{recall}/\text{precision} + \text{recall}$.

Kappa is another method of calculating classification accuracy to determine whether the predicted results are consistent with actual results, which is defined as $\kappa = \text{pr}(a) - \text{pr}(e)/1 - \text{pr}(e)$, where the pr(a) is the actual accuracy and pr(e) is the theoretical accuracy.

Receiver operating characteristic (ROC) curve is obtained by selecting the specificity (false-positive rate) and sensitivity (true-positive rate) as the horizontal axis and the vertical axis, respectively. The corresponding area under the receiver operating characteristic (ROC) curve (AUC) is one of the most popular metrics to probabilistically evaluate the performance of classifiers.

5.2. Results. In order to validate our detection method on malwares, we set two baselines for comparison: feature extraction and classification. The experiment performs malware detection task in three different ways on the same dataset, and the results are shown as Table 1. Obviously, the results

Table 1: The TP, TN, FP, precision, recall, accuracy, F1, and kappa of each malware detection method.

Method	TP	TN	FP	Precision	Recall	Accuracy (%)	F1	Kappa
A (CUCKOO + TF-IDF + SVM)	202	186	10	0.953	1	97.5	0.976	0.950
B (CUCKOO + GloVe + CNN)	203	191	4	0.981	1	98.9	0.990	0.980
C (API calls + GloVe + CNN)	193	180	15	0.928	0.951	93.8	0.939	0.87

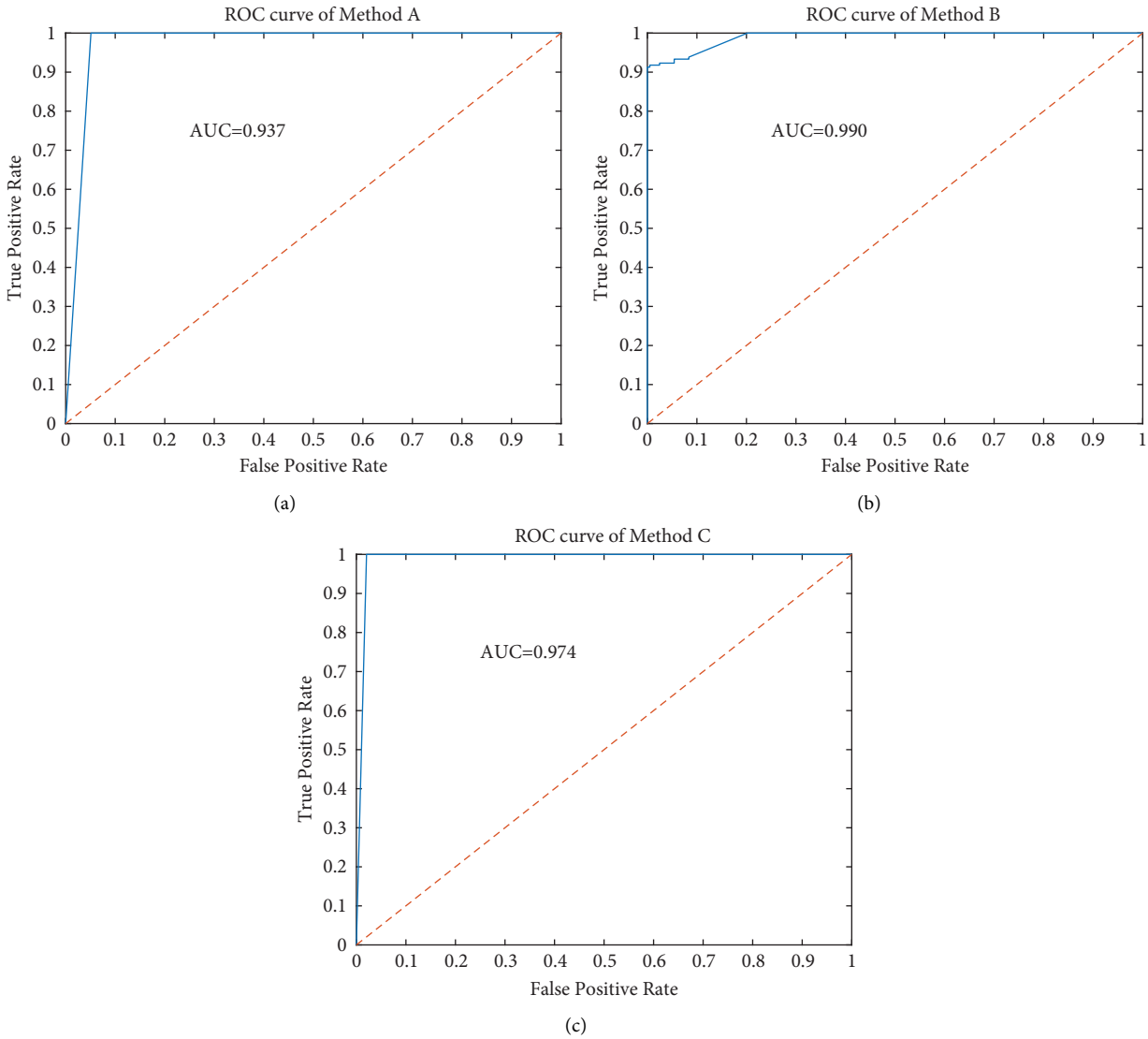

Figure 4: ROC curve of Method A (a), Method B (b), Method C (c).

illustrate that the malware detection method using CNN and Word Embedding outperformed both Method A and Method C in the accuracy, recall, F1, and kappa. Especially, in the aspect of Accuracy, Method A has achieved 98%, which is of great significance in practical for malware detection because the cost of misclassification is very high. As for Recall, the value of Method B is 1, which is equal to Method A . This is the only situation where our model performance is not over other models because its Recall has reached the optimal value 1. In addition, from Figure 4, we can observe that the AUC (area under the ROC curve) of Method B is 0.990, which is superior to 0.974 of Method A and 0.937 of Method B. This means that the model proposed in this article shows better classification ability. Next, we contrast our model with other methods in two stages: the feature extraction stage and the classification stage, respectively.

5.2.1. Feature Extraction with Word Embedding and API Calls Sequence. In the stage of feature extraction, Method C uses Cuckoo to extract the API call sequence of the program as description information for malware detection. Method C

divides the API calls sequence information into six categories (socket, memory management, processes, and threads etc.), which simplifies complexity of the feature extraction but also lost some other features. In our model (Method B), Cuckoo takes the natural language description information of program behavior, and GloVe represents the words as vectors remaining the global and local information. On the other hand, the Word Embedding has an inherent advantage in clustering so that the distribution of the similar words roughly similar in vector space. Therefore, it is reasonable that the accuracy of our model is higher than Method C.

5.2.2. Word Vector and TF-IDF. In the stage of classification, Method B and Method A choose different ways to analyze the data obtained previously. Considering that the TF-IDF used in Method A is a type of numerical data, we choose the SVM to complete the classification phase of Method A. Compared with the word embedding in Method A, TF-IDF requires a higher dimension of the space. On the other hand, the word vectors in Method B can express the semantic and grammatical information without removing the stop words because they also contain some grammatical information available for training. Therefore, using the word vectors can extract the features of program behavior more effectively and make the classification more accurate at the same time. The experimental results directly verify that the accuracy of Method B is higher than Method A.

6. Conclusion and Future Work

In this article, we proposed a novel approach for detecting malware using CNN via Word Embedding. Our approach first extracts the natural language description information of program behavior by Cuckoo Sandbox and uses GloVe to map the natural language (word space) into the corresponding vector space, which results in a dictionary of words represented by a real-valued vectors. For each program, the corresponding textual description information extracted by Cuckoo can be represented as a sequence of word vectors, called feature map. The task of malware detection can be equivalent to image classification, and we use the convolution neural network as a classifier to learn the feature maps of programs. In this way, we can make full use of the grammar and sematic information of programs. In the evaluation stage, we created two baselines to compare with our model, one method adopts TF-IDF word representation in the stage of features extraction (Method A) and uses a SVM as classifier in the stage of classification. The other method uses API calls sequences to represent the behavior information of executable program (Method C). The experimental results illustrate that our approach achieves a better performance than others.

Future work should focus on the defensive mechanisms that we identified as potentially helpful to improve our model. Also, the applicability of our detection model to additional domains should be studied. On the other hand, we expect to improve our means of extracting information so that we can generate more valuable features with more expressive ability for the behaviors of executable programs.

Data Availability

The experimental data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] S. Dai, Y. Liu, T. Wang, W. Tao, and Z. Wei, "Behavior-based malware detection on mobile phone," in *Proceedings of the International Conference on Wireless Communications Networking and Mobile Computing*, Niagara Falls, Canada, October 2010.

[2] I. Burguera, U. Zurutuza, and S. T. Nadjm, "Crowdroid:behavior-based malware detection system for android," in *Proceedings of the Acm Workshop on Security and Privacy in Smartphones and Mobile Devices*, Chicago, IL, USA, October 2011.

[3] A. Mujumdar, G. Masiwal, and B. B. Meshram, "Analysis of signature-based and behavior-based anti-malware approaches," *International Journal of Advanced Research in Computer Engineering and Technology*, vol. 2, no. 6, 2013.

[4] J. Pennington, R. Socher, and C. Manning, "Glove: global vectors for word representation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543, Doha, Qatar, October 2014.

[5] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of api calls," in *Proceedings of the Cybercrime and Trustworthy Computing Workshop*, pp. 52–59, Berlin, Germany, June 2010.

[6] G. G. Sundarkumar, V. Ravi, I. Nwogu, and V. Govindaraju, "Malware detection via api calls, topic models and machine learning," in *Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1212–1217, Gothenburg, Sweden, August 2015.

[7] O. Hachinyan, "Detection of malicious software on based on multiple equations of api-calls sequences," in *Proceedings of the 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 415–418, St. Petersburg and Moscow, Russia, February 2017.

[8] A. Pektas and T. Acarman, "Malware classification based on api calls and behaviour analysis," *IET Information Security*, vol. 12, no. 2, pp. 107–117, 2018.

[9] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 60, no. 1, pp. 493–502, 1972.

[10] R. K. Roul, O. R. Devanand, and S. K. Sahay, "Web document clustering and ranking using tf-idf based apriori approach," pp. 34–39, 2014, https://arxiv.org/abs/1406.5617.

[11] K. D. He, Z. T. Zhu, and Y. Cheng, "A research on text classification method based on improved TF-IDF algorithm," *Journal of Guangdong University of Technology*, vol. 33, no. 5, pp. 49–53, 2016.

[12] J. F. Zhang, "Words similarity algorithm based on improved TF-IDF and cosine theorem," *Modern Computer*, pp. 20–23, 2017.

[13] N. Kondylidis, M. Tzelepi, and A. Tefas, "Exploiting Tf-Idf In Deep Convolutional Neural Networks For Content Based Image Retrieval," *Multimedia Tools and Applications*, vol. 77, no. 23, pp. 1–20, 2018.