# DroidVecDeep: Android Malware Detection Based on Word2Vec and Deep Belief Network

**Tieming Chen[1], Qingyu Mao[1], Mingqi Lv[1]*, Hongbing Cheng[1], Yinglong Li[1]**
[1] College of Computer Science, Zhejiang University of Technology
Hangzhou, Zhejiang, 310023, China
[e-mail:{tmchen, mqyzjut, mingqilv*,chenghb,liyinglong }@zjut.edu.cn]
*Corresponding author: Mingqi Lv

*Received July 30, 2018; revised October 24, 2018; accepted November 5, 2018; published April 30, 2019*

## Abstract

With the proliferation of the Android malicious applications, malware becomes more capable of hiding or confusing its malicious intent through the use of code obfuscation, which has significantly weaken the effectiveness of the conventional defense mechanisms. Therefore, in order to effectively detect unknown malicious applications on the Android platform, we propose *DroidVecDeep*, an Android malware detection method using deep learning technique. First, we extract various features and rank them using Mean Decrease Impurity. Second, we transform the features into compact vectors based on word2vec. Finally, we train the classifier based on deep learning model. A comprehensive experimental study on a real sample collection was performed to compare various malware detection approaches. Experimental results demonstrate that the proposed method outperforms other Android malware detection techniques.

***Keywords:*** Android security, malware detection, deep learning, distributed representation, word2vec

This paper is partially supported by the key national Natural Science Foundation of China with Grant No. 61772026, No. U1509214, No.61602412 and No. 61502421, the Zhejiang Provincial Natural Science Foundation of China with Grant No. LY18F020033, No. LY19F020083.

**http://doi.org/10.3837/tiis.2019.04.025** **ISSN : 1976-7277**

## 1. Introduction

With the development of the mobile Internet, smartphone shipments grow rapidly, especially the domestic manufacturers of smartphones for the Android system. Android, as an open source and customizable operating system for smartphones, has more risks. According to the 2017 CVE(Common Vulnerabilities and Exposures) details report [1,2], the Android system ranked 842 holes among the number of product vulnerabilities, which is increased by 61.0% as compared with that of 2016. Malicious software attack methods are diversified, and the distribution channels are multi-polarized, making it difficult to detect and prevent. As a result, manufacturers have a long delay in the repair of vulnerabilities. Faced with this severe challenge, we need better methods for malicious detection.

Recently, a lot of researches have focused on signature-based detection methods, i.e., using static or dynamic analysis to discover high recognizable patterns [3,4,5], which are then used to characterize malware. However, this kind of methods would become less effective when detecting unknown malware. It's more difficult to detect malicious applications using traditional methods due to code obfuscation, transformation attacks, etc. Moreover, during our analysis of Android applications for further studies, we have discovered that some of the benign applications also possess seemingly malicious characteristics, which makes them difficult to distinguish. Therefore, some recent researches tried to use machine learning techniques [6,7,8,9,10] to detect unknown Android malware. These works extract features using static or dynamic analysis and learn the difference between benign and malicious applications automatically.

Deep learning, which is part of a broader family of machine learning methods based on learning data representations, has gained increasing attention in many fields. Deep learning architectures overcome the learning difficulty through stratified training, such as deep belief network [30], which pre-train multiple layers from bottom to up to construct the classification model. In this paper, we focus on the utilization of deep neural networks on feature representation to improve the detection performance. We propose *DroidVecDeep*, a novel Android malware detection method using deep learning technique. First, we extract various features from Android apps. Second, we transform these features into a high-level representation based on word2vec. Finally, we feed them into a deep learning model to build the classifier. Experiments on real-world apps show that deep learning and word embedding are suitable for characterizing Android malware features and improve detection accuracy. We make the following contributions:

(1) We design *DroidVecDeep*, an automatic Android malware detection system, which combines static analysis and deep learning.

(2) We extract various features and rank them by mean decrease impurity in random forest, which reduce feature dimensions and make the system more lightweight. Moreover, we transform the features into high-level representation using word2vec, which can then better characterize Android malware. According to the experiment, there is a 1-2% improvement in classification accuracy.

The remainder of this paper is organized as follows. Related works are described in Section 2. Section 3 briefly describes *DroidVecDeep* and its design. Section 4 introduces the details of feature extraction and deep learning model. The experiments we conducted and analysis are

described in Section 5.Finally, we draw our conclusions in Section 6.

## 2. Related Work

### Feature Representation on Android Malware Detection

Drebin[11] uses static analysis to extract as many application features as possible (such as permissions, API calls, network addresses, etc.) to characterize malicious applications; Maldetect[12] extracts Dalvik instructions from dex files and simplify them by symbolizing opcode. Then, N-gram encoding of the instruction sequence is used as the input feature of the classification; DroidSieve [13] extracts massive features centered on resources and semantics, and sorts the features to find the core features; FrequenSel [14]proposes a feature selection algorithm based on the frequency difference between the malicious application and benign application; HinDroid[15] extracts the API to construct a structured heterogeneous information network and characterized the relationship between APIs. The depth analysis and characterization of the extracted features make the classification better than the traditional detection methods.

Yi Luan et al. [16] uses the word vector as the input layer of the neural network in the task of oral comprehension to alleviate the problem of overtraining; Edward Raff et al. [17] converts the bytes in the PE header into the word vector through word2vec, then map features to high-dimensional space for input layers of deep neural networks. A number of research studies have shown that transforming features into word vectors is used as an input layer of a deep neural network, which has a better application for classification tasks.

### Android Malware Detection using Deep Learning

HinDroid[15] extracts APIs to build a structured heterogeneous information network, extracts multiple meta-paths, integrates different similarity methods using a multi-core architecture, and builds a classifier with better recognition capabilities; MalDozer[18]based on an artificial neural network that takes, as input, the raw sequences of API method calls, as they appear in the dex file, to enable malware detection and family attribution.; Deep4MalDroid[19] extracts Linux system kernel calls, constructs weighted directed graphs and uses deep confidence networks for malicious classification.

DroidDetector[20] extracts Android applications Permissions, sensitive APIs, and dynamic behavior data, then firstly apply them into DBN, a deep learning model and achieved good results. DroidDeep[21] extracts more than 30,000 multi-level features and combines both static analysis and deep learning that is capable of detecting Android malware with a high accuracy and a low false alarm rate.

The above work extracts and selects features from various perspectives, using a variety of deep learning models for modeling and classification. However, the treatment of features in some research work is relatively simple. The use of feature word embeddings combined with deep neural network has a good application in processing natural language and malicious detection in the PC field and has not yet been applied to mobile security detection.

## 3. System Architecture

In this section, we present the architecture of *DroidVecDeep*, as shown in **Fig. 1**, which consists of the following components.

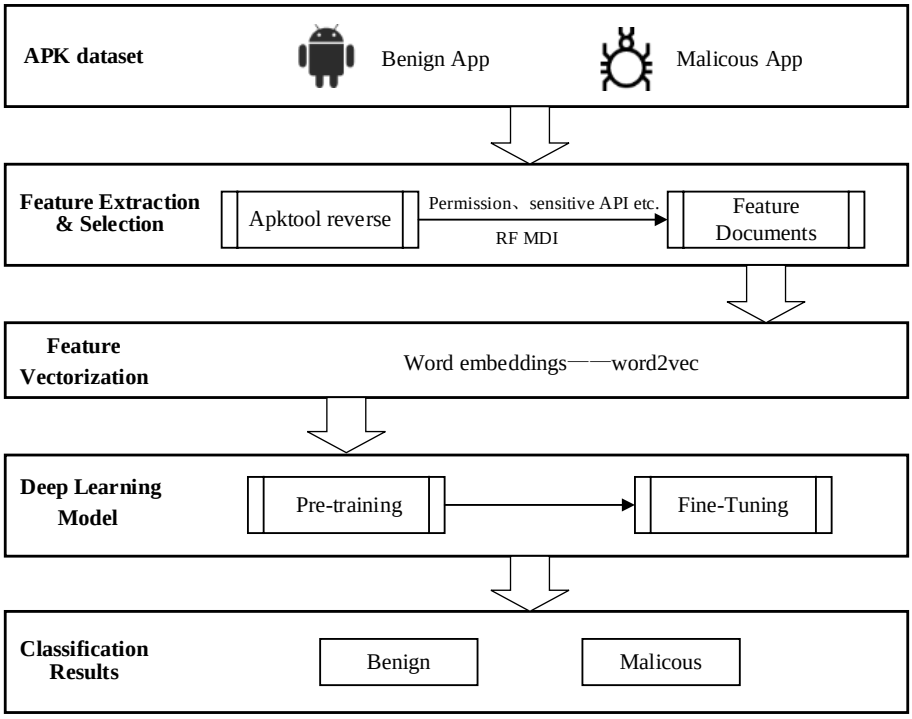



**Fig. 1.** Architecture of *DroidVecDeep*

(1) *APK dataset*: We establish a dataset with 15,000 Android applications. It contains malicious applications of the most recent two months from Drebin[11], VirusTotal[22], and Contagio[23]. Since it is impossible to directly download APK from Google Play[32], so the benign samples are mainly downloaded from APKPure[24] and the domestic 360 application market[25]. To ensure the downloaded apks are benign applications, we download the most popular150 applications from 20 categories in the two application markets ranked by download counts.

(2) *Feature extraction & selection*: We use static techniques to analyze applications. We extract features such as permissions, actions, and sensitive API calls, and use random forests for feature selection (See Section 4.1 for details).

(3) *Feature vectorization*: We model the extracted features as a document, and then use word2vec to analyze the documents and transform the features into K-dimensional word vectors (See Section 4.2 for details).

(4) *Malicious classification*: We use the DBN (Deep Belief Networks) model to establish an optimal detection classifier for Android application classification (See Section 4.3 for details).

## 4. Feature Extraction and Classification

### 4.1 Feature Extraction and Selection

To systematically characterize Android apps, we conduct static analysis to extract four types of features. We reverse the APK and extract the AndroidManifest.xml file and the folder containing the smali source code with Apktool [26]. Then, we perform feature extraction as

follows:

(1) We extract Permission and Intent Action from AndroidManifest.xml. Permission describes the permissions required by the application. Some of them would be potentially risky. For example, *android.permission.ACCESS_WIFI_STATE* is the permission to allow access to WIFI status, *android.permission.SEND_SMS* is the permission to allow send sms, *android.permission.CALL_PHONE* is the permission to allow make a phone call. In this step, we looked for a total of 120 risky permissions. We also scan the code to identify any explicit intents, which are used to start services within the same app. Action in Intent describes the common actions of the application, which is an important feature to represent malicious behavior. For example, *android.intent.action.ANSWER* is the intent to allow hand incoming calls, *android.intent.action.CAMERA_BUTTON* is the intent to allow take photos.

(2) Java code is compiled to generate dex file, which can be run in the Android Dalvik virtual machine. The smali code is Dalvik's disassembly language, which can be obtained by decompiling dex files. Here, since there are a large number of common API calls shared between apps, we tend to analyze only sensitive APIs and use them for detection. For example, *getDeviceId()* can get the device information, and *sendTextMessage()* can send SMS. These methods have sufficient intent to reveal user privacy data or perform certain malicious operations implicitly. In this step, we analyze the smali code and extract more than 40 sensitive APIs as features. Some sensitive APIs are shown in **Table 1**. The risk level is based on the frequency of the API call by the malicious application.

(3) We extract some sensitive strings from the smali source code as features. For example, *ro.serialno* indicates the unique device number, and *sms_body* is related to the text message.

**Table 1.** Sensitive APIs

| Sensitive APIs | Type | Risk Level | Sensitive APIs | Type | Risk Level |
|---|---|---|---|---|---|
| getDeviceId | Privacy | High | getNetworkOperator | Privacy | High |
| getSubscriberId | Privacy | High | getSimSerialNumber | Privacy | High |
| getLine1Number | Privacy | High | getAccounts | Privacy | High |
| getInstalledApplications | Privacy | Medium | getRunningAppProcesses | Privacy | Low |
| getRunningTasks | Privacy | Low | getPhoneType | Privacy | Low |
| getNetworkOperatorName | Privacy | Low | getAccountsByType | Privacy | Low |
| getLastKnownLocation | Location | High | getAllCellInfo | Location | High |
| getNeighboringCellInfo | Location | High | getCellLocation | Location | High |
| startDiscovery | Devices | High | setPreviewDisplay | Devices | High |
| getBondedDevices | Devices | High | MediaRecorder | Devices | High |
| setWifiEnabled | Devices | High | takePicture | Devices | Medium |
| startPreview | Devices | Medium | sendTextMessage | Control | High |
| abortBroadcast | Control | High | sendMultipartTextMessage | Control | High |
| createFromPdu | Control | High | SendDataMessage | Control | High |
| requestLocationUpdates | Control | High | execHttpRequest | Control | Medium |

For each sample, we extract features based on the above steps and combine them as a feature vector. The preliminary extracted features are shown in **Fig. 2**. On the basis of these features, we conduct feature selection based on the Mean Decrease Impurity (MDI) importance [27] of random forests.

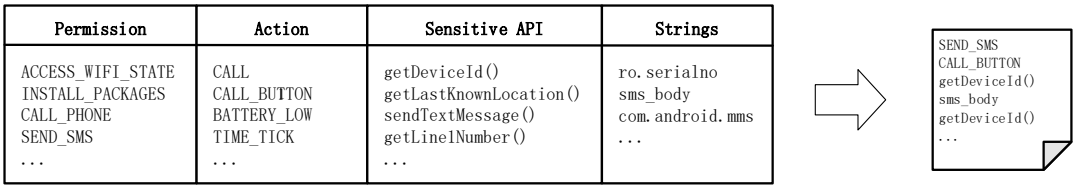



**Fig. 2.** Features

A classification tree is an input-output model represented by a tree structure $T$, from a random input vector $(X_1, \ldots X_p)$ taking its values in $\chi_1 \times \ldots \times \chi_p = \chi$ to a random output variable $Y \epsilon \gamma$. Any node $t$ in the tree represents a subset of the space, with the root node being $\chi$ itself. Internal nodes $t$ are labeled with a binary test (or split) $S_t = (X_m < c)$ dividing their subset in two subsets corresponding to their two children $t_L$ and $t_R$ ($c$ is a constant for subsets partitioning), while the terminal nodes (or leaves) are labeled with a best guess value of the output variable. A tree is built from a learning sample of size N drawn from $P(X_1, \ldots X_p, Y)$ using a recursive procedure which identifies at each node $t$ the split $S_t = s *$ for which the partition of the $N_t$ node samples into $t_L$ and $t_R$ maximizes the decrease of some impurity measure $i(t)$ (e.g., the Gini index, the Shannon entropy, or the variance of $Y$)

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R) \tag{1}$$

And where $p_L = N_{t_L}/N_t$ and $p_R = N_{t_R}/N_t$.

$$\mathrm{Imp}(X_m) = \frac{1}{N_T} \sum_{T} \sum_{t \in T: v(s_t) = X_m} p(t) \Delta i(s_t, t) \tag{2}$$

A random forests consists of multiple classification trees. Each node in the decision tree is a condition about a feature in order to divide the data set into multiple copies according to different features. The node (optimal condition) can be determined using the impurity, here we use the Gini impurity, which is represented by Equation 2. When training the decision tree, we can calculate how many trees are reduced in impurity for each feature. For a decision tree forests, the average reduction in the impurity of each feature can be calculated and the average reduced impurity is used as the value of the feature selection. Through the value selection, features of higher importance are extracted as features of the final Android malicious classification.

We perform feature selection as follows: First, We analyze apks and extract feature according to our feature list. Then, we transform the feature set into vector $V = \{0,1,0,0,1,..\}$, in which 1 indicates that the feature is contained in this app, whereas 0 indicates not. Finally, we use random forests to build a model and get the ordered features list ranking by MDI.

### 4.2 Feature Vectorization

According to Section 4.1, the extracted features are represented as a feature vector based on one-hot encoding. According to the researches in the NLP domain, one-hot encoding does not contain any corpus information, and the distance between all words is the same. Word2vec[28]

defines the vector of words according to the context, and the words with high relevance have closer distances. That is to say, word2vec is more expressive and more capable of expressing the intrinsic characteristics of data. Word2vec can utilize either of two model architectures to produce a distributed representation of words: the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. In the skip-gram architecture, the model uses the current word to predict the surrounding window of context words. Joshua Saxe et al. [29] proposed a method to distinguish malware with neural networks. In their research, 4 types of features (Byte/Entropy Histogram, PE Import, String 2D histogram, PE Metadata) are converted into 256-dimensional vectors one by one manually. Their system achieves a 95% detection rate at 0.1% false positive rate, based on more than 400,000 software binaries.

Inspired by Joshua Saxe's method and NLP technique, we try to apply word2vec to Android malware classification for features representation with word embeddings. For our case, we treat features extracted from apk files as words. In detail, each feature is represented as a k-dimensional vector. We use CBOW model for training. The final trained matrix is $N \times (K \times X)$ dimensions, where $N$ is the number of samples, $K$ is the word vector dimension, and $X$ is the number of features. Each vector represents a point in the k-dimensional space, and each element of the vector is determined by repeatedly training and adjusting the weight for the features. We use $K=100$ in all models.

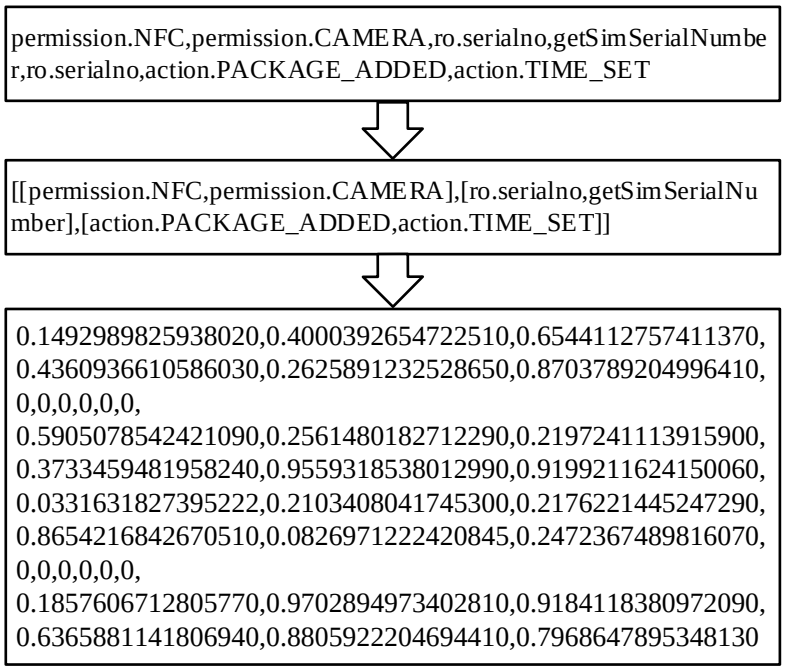



**Fig. 3.** Feature Vectorization Process

The specific vectorization process is described in algorithm1 and **Fig. 3**. (1) In lines 3-4, we perform feature selection according to the above description. (2) In lines 5, we divide feature document into four sentences according to the category of the feature. (3) In lines 6-7, we use the word2vec model for training to obtain the word vector. (4)In lines 8-13, if some features are not included in the sample, we fill it with 0. We simply describe the results of algorithm 1 in

**Fig. 3**. For convenience, we set *k*=6 in **Fig. 3**. The first part is the original feature list (Assume that the sample contains only six features.). The second part is the result of feature dividing (Algorithm1 step2). The last part is the result of feature vectorization (Algorithm1 step3-4). For example, if a sample contains *permission.NFC*, we can obtain word vector trained by word2vec, which is [0.1492989825938020,0.4000392654722510,0.6544112757411370,0.43609366105860-30,0.2625891232528650,0.8703789204996410]. If a sample does not contains *permission.CAM-ERA*, the vector is [0,0,0,0,0,0].

**Algorithm1：Feature Vectorization**

**Input：**
D: sample features documents
K: dimension of word2vec
F: feature list
**Output:**
S: N ×(K × X)-dimension vector

```
1  begin
2    foreach Di in D do
3      sentences = empty_list();
4      Di = feature_select(Di);
5      sentences ← three sentences extract from Di;
6      model = train_word2vec(sentences,K);
7      word_dict←model.wv.vocab;
8      zero_vec← K-dimension zero vector;
9      foreach Fi in F do
10       if Fi in wordDict do
11         S.extend(wordDict[Fi]);
12       else
13         S.extend(zero_vec);
14   return S
15 end
```

### 4.3 Deep Learning Model

Deep neural networks have a variety of structures, such as deep belief networks, convolutional neural networks, etc. They all have multiple hidden layers. The deep belief network [30,31] is a fast, greedy learning algorithm, which is able to learn typical features and has a good effect on the processing of one-dimensional data. Finally, we choose the deep belief network and softmax classifier to characterize and classify Android applications.

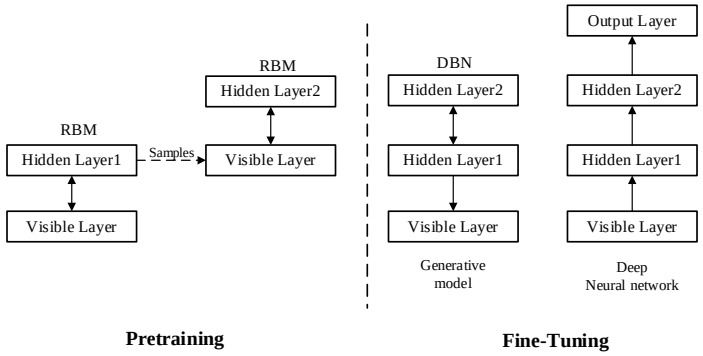



**Fig. 4.** DBN Training Process

The Deep Belief Network (DBN) can be seen as a stack of multiple constrained Boltzmann machines. The hidden layer of each restricted Boltzmann machine is viewed as the upper layer of the restricted Boltzmann machine. Furthermore, the deep belief network can be trained quickly by layer-by-layer training that is, starting from the lowest level, training only one layer at a time until the last layer. The deep confidence network training process can be divided into two stages: pre-training and fine-tuning. Firstly, the parameters of the model are initialized to better values through layer-by-layer pre-training, and then the parameters are fine-tuned through traditional learning methods. The training process as shown in **Fig. 4**.

**Algorithm2：DBN Greedy Learning**

**Input：**

Training Set: $\widehat{\boldsymbol{v}}^{(\boldsymbol{n})}, \boldsymbol{n} = \mathbf{1}, \ldots, \boldsymbol{N}$;
Learning Rate: $\boldsymbol{\alpha}$
DBN Layer: **L**
No. l weights: $\boldsymbol{w}^{(\boldsymbol{l})}$
No. l threshold: $\boldsymbol{a}^{(\boldsymbol{l})}$
No. l threshold: $\boldsymbol{b}^{(\boldsymbol{l})}$

**Output:**

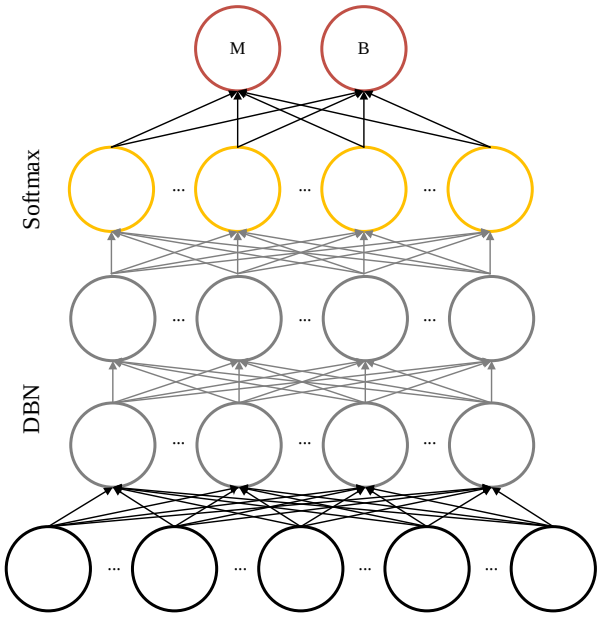
Weights: $\boldsymbol{W_1}, \ldots \boldsymbol{W_l}$

1 **begin**
2 **for** $\mathbf{l} = \mathbf{1} \ldots \mathbf{L}$ **do**
3 Initialization：$\boldsymbol{w}^{(\boldsymbol{l})} \leftarrow \mathbf{0}, \boldsymbol{a}^{(\boldsymbol{l})} \leftarrow \mathbf{0}, \boldsymbol{b}^{(\boldsymbol{l})} \leftarrow \mathbf{0}$;
4 **for** $\mathbf{i} = \mathbf{1} \ldots \mathbf{l} - \mathbf{1}$ **do**
5 Sample $\boldsymbol{h}^{(\boldsymbol{i})}$ according to $\mathbf{q}\left(\boldsymbol{h}^{(\boldsymbol{i})} \middle| \boldsymbol{h}^{(\boldsymbol{i}-\mathbf{1})}\right)$;
6 Use $\boldsymbol{h}^{(\boldsymbol{l}-\mathbf{1})}$as a training sample to fully train the Lth limited Boltzmann machine $\boldsymbol{w}^{(\boldsymbol{l})}, \boldsymbol{a}^{(\boldsymbol{l})}, \boldsymbol{b}^{(\boldsymbol{l})}$
7 **end**

During the pre-training process, the layer-by-layer training method is used to simplify the training of the DBN in the training of multiple restricted Boltzmann machines. The specific training process is described in Algorithm 2 (lines 2-6). A lot of practice shows that pre-training can produce very good initial values of parameters, which greatly reduces the difficulty of learning the model.

In the fine-tuning stage, after the pre-training, combined with the Android malicious application classification task of this article, the global learning algorithm can be used to fine tune the parameters, and the model is converged to a better local optimum.

In this study, the deep belief network is applied to detection of Android malicious applications. The system model diagram is shown in **Fig. 5**. The transformed (K × X)-dimensional word vector is input into the deep belief network for pre-training to provide the initial weight of the neural network. Then, fine-tuning of the parameters is performed, and a softmax classifier is added at the top layer to be used as a classification of malicious applications, and the classification result is finally output.




**Note**:M:malicious B:benign

**Fig. 5.** Android malware detection model

## 5. Evaluations

In this section, we proceed to an empirical evaluation to fully evaluate the performance of the proposed Android malware detection system. The experiment uses the most common model evaluation metrics (i.e., Precision, Recall, F1) for machine learning. In particular, we mainly conduct the following three experiments:

(1) **Detection performance**. We evaluated the detection performance of *DroidVecDeep* and other popular classification models with word embedding or not in the same dataset.

(2) **Features exploitation**. We performed an in-depth analysis on the effectiveness of the extracted features.

(3) **Related works comparison**. We compare the detection results of the *DroidVecDeep* with the related work mentioned above.

### 5.1 Experimental Setup

The test dataset contains 15,000 applications. Malicious applications were malicious samples from VirusTotal 2017-10 and 2018-3 and Drebin datasets, totaling 12,000. Benign applications were obtained from the APKPure and 360 markets, totaling 3,000. We crawled according to the order of the leaderboards. Due to imbalanced samples, both the training set and the test set were randomly selected from among them, with a total of 3,000 samples and a sample ratio of 1:1. The 10-fold cross-validations are conducted for the evaluation.

All the experiments were performed under the following environment: Intel Core i5-7500 @3.4GHz CPU, 8GB RAM, GeForce GTX1050 GPU; the software environment is as follows: cuda9.0, cudnn9.0, TensorFlow-gpu1.7. The algorithm is mainly implemented in the python language.

The most common evaluation metrics include true positive (TP), false positive (FP), true negative (TN), and false negative (FN). These four metrics can make up a confusion matrix as shown in **Table 2**.

**Table 2.** Confusion matrix

| Prediction | Malicious | Benign |
|---|---|---|
| Malicious | *TP* | *FN* |
| Benign | *FP* | *TN* |

Depending on these basic metrics, a series of common evaluation metrics can also be generated as follows.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times Recall \times Precision}{recall + precision}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the all observations in actual class F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost.

### 5.2 Comparisons of Word Embeddings in Classification Methods

Experiments are divided into two groups. One group generates feature vectors based on one-hot encoding. The other group uses word2vec to convert original features into semantic feature vectors. Finally, we use DBN, C4.5, SVM and Naive Bayes algorithm as classifiers for malicious classification and evaluate their performance. To ensure that word vectors can be generated for each feature, we set min_count=1, where min_count is a parameter in word2vec. When the frequency of the word in the text is lower than min_count, it will be automatically ignored and the corresponding word vector will not be generated. The experimental results are shown in **Table 3**.

**Table 3.** Comparisons of Word Embeddings in Typical Classification Methods

| Classifier | Malicious (%) | | | Benign (%) | | | Precision(%) |
|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Precision | Recall | F1 | |
| DBN | 97.34 | 97.17 | 97.25 | 97.24 | 97.41 | 97.32 | 97.29 |
| DroidVecDeep | **99.10** | **99.40** | **99.25** | **99.40** | **99.30** | **99.35** | **99.25** |
| C4.5 | 93.50 | 96.80 | 95.10 | 96.70 | 93.40 | 95.00 | 95.10 |
| w2v-C4.5 | 97.60 | 95.30 | 96.44 | 97.80 | 98.30 | 98.05 | 97.50 |
| SVM | 94.20 | 98.70 | 96.40 | 98.60 | 94.10 | 96.30 | 96.40 |
| w2v -SVM | 98.80 | 97.30 | 98.04 | 98.50 | 98.30 | 98.40 | 98.65 |
| NB | 89.90 | 94.90 | 92.30 | 94.70 | 89.60 | 92.10 | 92.30 |
| w2v -NB | 98.20 | 97.30 | 97.75 | 99.10 | 98.30 | 98.70 | 98.65 |

According to the results shown in **Table 3**, the performance of SVM is far better than the other two algorithms, so it is often used as the preferred classifier in malicious detection. Although the overall Precision of SVM is higher, the recognition rate for malicious software is much lower than the benign recognition rate. As a deep learning classification algorithm, DBN can learn features better. Regardless of the precision or recall, it has a certain improvement compared to traditional machine learning algorithms. In the representation of features, the result of the word vector is also superior to the feature vector generated based on one-hot encoding. This shows that it is not sufficient to characterize the application based on the existence of the feature. The feature is passed through the shallow neural network word2vec. After training, it can better characterize the frequency of features in the application, which facilitates further classification work.

### 5.3 Features Exploitation

Compared with the current detection methods of the same kind, some methods choose all Android permissions and components as features, but only a few applications contain so many permissions in reality. The addition of such features does not have a great impact on the results of the classifier. Instead, it increases the computational complexity of processing and degenerates the efficiency of classifier learning. Our study uses random forests for feature selecting and dimensionality reduction, which optimizes the performance of the system to some extent.

This section analyzes the feature selection strategy based on Mean Decrease Impurity in random forests. The decrease in the average impurity indicates the average degree of reduction of each feature to the error, which is characterized by Gini impurity. The top-10 permission, sensitive API and action features are shown in **Fig. 6-8**. Malware has a very large proportion of ransomware. Therefore, *SYSTEM_ALERT_WINDOW* permission is used to popup window, *WRITE_SMS, SEND_SMS* permission to send SMS messages are required by the malicious software. Corresponding to the permissions, *sendTextMessage()* is also the most important API in the sensitive API, and the rest are all API calls to get sensitive data from mobile phones. An intent is a description of an operation to be performed and an action indicates a general operation. Corresponding to the above description, *ACTION.SEND* can deliver some data to someone else, *ACTION.CALL* can perform a call to someone specified by the data. Sensitive strings always contains some privacy data, such as *ro.serialno, sms_body*. Since the number of features of sensitive strings is small, we have not performed feature selection on them.

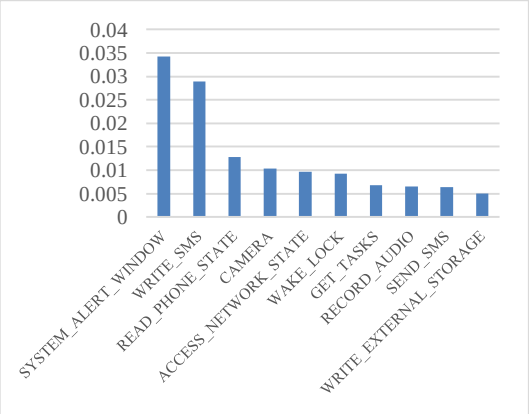



**Fig. 6.** Top-ranked permission features

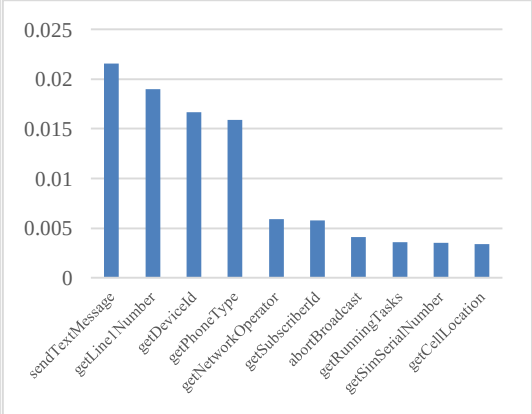



**Fig. 7.** Top-ranked sensitive API features

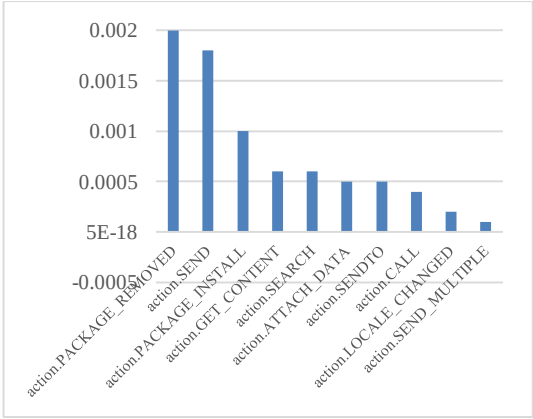



**Fig. 8.**Top-ranked action features

## 5.4 Parameter Tuning

This section tunes the parameters of *DroidVecDeep*, which employs a DBN as the classifier. The parameters includes the number of pre-training iterations, the number of fine-tuning iterations, the number of hidden layers, the number of nodes, and the mini-batch.

**Table 4** shows the impact of two parameters (number of hidden layers and their number of nodes) that are important to the DBN on the classification result. Because it is an in-depth neural network model, unlike shallow networks such as RBM, the experiment sets the hidden layer parameter from layer 2 to layer 5, and compares the classification results. According to the experimental results, it can be concluded that when the hidden layer number is 2 and the number of nodes at each layer is 200, the classification result is optimal, and the precision is 97.29%.

**Table 4.** Comparisons between different deep learning model constructions

| Num of Neurons | Malicious (%) | | | Benign (%) | | | Precision(%) |
|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Precision | Recall | F1 | |
| [150,150] | 97.25 | 97.12 | 97.19 | 97.19 | 97.32 | 97.26 | 97.22 |
| [200,200] | **97.34** | 97.17 | **97.25** | **97.24** | **97.41** | **97.32** | **97.29** |
| [250,250] | 97.21 | 97.12 | 97.16 | 97.19 | 97.28 | 97.23 | 97.20 |
| [150,150,150] | 96.95 | 97.30 | 97.13 | 97.36 | 97.01 | 97.18 | 97.16 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| [200,200,200] | 96.86 | 97.21 | 97.04 | 97.27 | 96.93 | 97.10 | 97.07 |
| [250,250,250] | 96.82 | **97.35** | 97.08 | 97.40 | 96.88 | 97.14 | 97.11 |
| [200,200,200,200] | 97.16 | 97.08 | 97.12 | 97.15 | 97.23 | 97.19 | 97.16 |
| [200,200,200,200,200] | 97.42 | 96.63 | 97.02 | 96.73 | 97.50 | 97.11 | 97.07 |

**Note:** Num of Neurons: The number of neurons. [150,150,150] means that the DBN network contains 3 layers and each layer contains 150 neurons.

### 5.5 Comparisons of Sample Size

This section mainly focuses on the number of samples for experimental analysis. We randomly selected 500, 1,000, 2,000, and 3,000 samples from the APK dataset for multiple experiments. The proportion of malicious and benign samples was 1:1. The experimental results are shown in **Table 5**. The comprehensive evaluation index of sample size is better than that of small sample size. The larger the sample size, the better the overall performance is.

**Table 5.** Comparisons between different numbers of samples

| number of samples | Malicious (%) | | | Benign (%) | | | Precision (%) |
|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Precision | Recall | F1 | |
| 500 | 97.66 | 94.70 | 96.15 | 94.26 | 97.46 | 95.83 | 96.05 |
| 1000 | 96.91 | 97.67 | 97.29 | 97.51 | 96.71 | 97.11 | 97.20 |
| 2000 | **98.68** | **98.12** | **98.40** | 97.88 | **98.50** | **98.19** | **98.30** |
| 3000 | 97.84 | 97.98 | 97.91 | **98.02** | 97.89 | 97.96 | 97.93 |

### 5.6 Comparisons with Baselines

In order to evaluate the effectiveness of the *DroidVecDeep*, we compare the proposed method with several baselines. *Drebin*[11]extracts static features and uses SVM model to classify Android malware. *DroidDeep*[21]extracts API call, permission, component etc. And it uses DBN to build a classifier. We extract the corresponding number of features as much as possible for experiment and use the same dataset from Section 5.1 to train *Drebin*, *DroidDeep* and our own model respectively. The results are shown in **Fig. 9**.

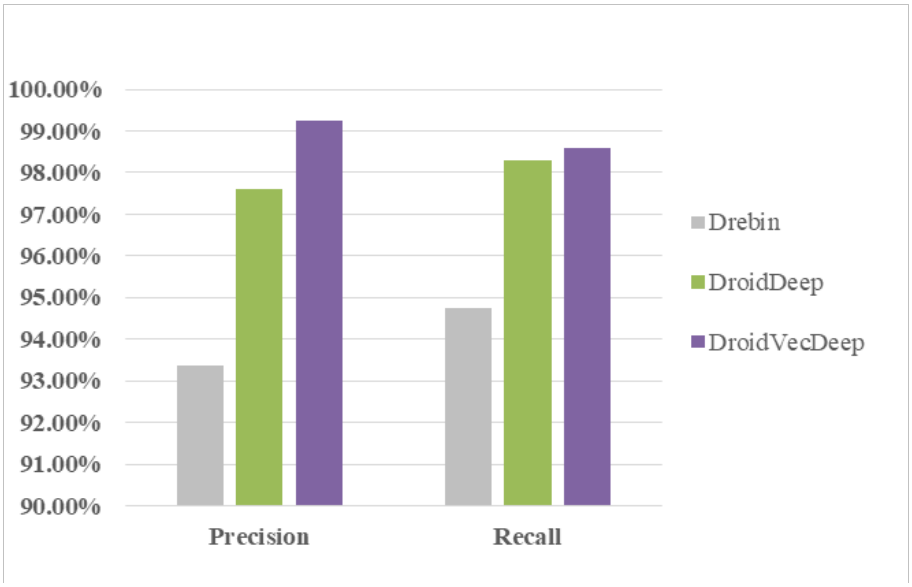



**Fig. 9.** Comparisons with Baselines

We can see that our method performs better than the other two baselines, achieve 99.1% accuracy. *Drebin* and *DroidDeep* only achieve 93.37% and 97.6% respectively. The experimental results clearly show that *DroidVecDeep* is better than the state-of-the-art machine learning algorithms.

On the other hand, *DroidDeep* extracts the features and construct vectors based on one-hot encoding, at the same time, *DroidVecDeep* uses word2vec to transform features into the word embeddings, which compensates for the disadvantages of insufficient feature frequency characterization. Under the same classifier, it is obvious that our method has more advantages in feature transformation and expression.

### 5.7 Comparisons with antivirus scanners

In this section, we compare *DroidVecDeep* with several off-the-shelf antivirus scanners. For this experiment, we randomly select 1000 samples from our dataset, containing 500 malicious samples and 500 benign samples. To compete with common antivirus products, we send each sample to our trained model and VirusTotal [22] platform respectively. VirusTotal integrates a variety of antivirus engines, thus we can get the detection result from 10 well-known antivirus scanners (Avira, ClamAV, Comodo, ESET, Kaspersky, Kingsoft, McAfee, 360, Symantec, Tencent). Finally, we obtain the detection rates and false positive rates by the statistics of output.

**Table 6.** Comparisons with antivirus scanners

| Result (%) | avs1 | avs2 | avs3 | avs4 | avs5 | avs6 | avs7 | avs8 | avs9 | avs10 | Our |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Precision | 98.98 | 99.48 | 98.68 | 97.83 | **99.59** | 65.56 | 97.27 | 98.93 | 99.13 | 97.18 | 99.21 |
| Recall | 97.40 | 77.13 | 15.00 | 99.20 | 97.87 | 0.47 | **99.60** | 92.80 | 91.67 | 96.47 | 98.87 |
| F1 | 98.18 | 86.89 | 26.04 | 98.51 | 98.72 | 0.93 | 98.42 | 95.77 | 95.25 | 96.82 | **99.04** |

**Note: avs1-avs10: Avira, ClamAV, Comodo, ESET, Kaspersky, Kingsoft, McAfee, 360, Symantec, Tencent**

The results of the experiments are shown in **Table 6**, most antivirus scanner has a detection rate of over 95%, while there are some scanners that detection rates are below 70%. Obviouslythese antivirus scanners may not be specialized in detecting mobile applications. Among them, our detection method ranks third here, and obtain a detection rate of 99.21%. In addition, these samples have been public for a longer time, thus almost all antivirus scanners have the signature of malicious samples. The deep learning method has much more strengths than the traditional technique when the samples are unknown malware.

From the perspective of recall rate, most antivirus scanners have a higher recall rate, but, for example, ClamAV, which achieves high accuracy and low recall rate, it may be that the mobile signatures database update untimely. And *DroidVecDeep* achieves a high recall rate. Weight both precision and recall, we get 99.04% F1 score.

## 6. Conclusion

In this paper, we propose *DroidVecDeep*, an Android malware detection method using deep learning based on word2vec embeddings. In this work, we firstly extracted a total of 240 features from 4 main static feature types of Android apps, and then use word embeddings for characterization. Finally, we use a DBN-based deep learning to build the classifier. Our work compensates for the lack of extraction and characterization of some features in relevant work. We evaluate it with 3000 benign apps and 12000 malware in real life. Experimental results

show that *DroidVecDeep* performs well in accuracy and execution efficiency and is superior to some malicious detection tools.

## References


[1] IDC: Smartphone OS Market Share. https://www.idc.com/promo/smartphone-market -share/os.

[2] 2017 Special Report on Android Malware. http://blogs.360.cn/360mobile/2018/03/01/ review_Android_malware_of_2017/.

[3] A. Shabtai, Y. Fledel, U. Kanonov et al., "Google android:a state-of-the-art review of security mechanisms," 2009. Article (CrossRef Link).

[4] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital investigation*, vol. 13, pp. 22-37, 2015. Article (CrossRef Link).

[5] A.P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM*, pp. 3-14, 2011. Article (CrossRef Link).

[6] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638-646, 2018. Article (CrossRef Link).

[7] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant Permission Identification for Machine Learning Based Android Malware Detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216-3225, 2018. Article (CrossRef Link).

[8] C. Wang, Z. Li, X. Mo, H. Yang, and Y. Zhao, "An android malware dynamic detection method based on service call co-occurrence matrices," *Annals of Telecommunications*, vol. 72, pp. 607-615, 2017. Article (CrossRef Link).

[9] Y. Xu, C. Wu, K. Zheng, X. Niu, and T. Lu, "Feature Selection to Mine Joint Features from High-dimension Space for Android Malware Detection," *KSII Transactions on Internet & Information Systems*, vol. 11, no. 9, pp.4658-4679, 2017. Article (CrossRef Link).

[10] T. Chen, X. Zhang, S. Jin, and O. Kim, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications*, vol. 41, pp. 5972-5983, 2014. Article (CrossRef Link).

[11] D. Arp, M. Spreitzenbarth, M. H¨ubner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proc. of 21st Annual Network and Distributed System Security Symposium (NDSS'14)*, pp. 1–15, San Diego, CA, USA, February 2014. Article (CrossRef Link).

[12] T. Chen, Y. Yang et al., "Maldetect: An Android Malware Detection System Based on Abstraction of Dalvik Instructions," *Journal of Computer Research and Development*,vol. 53, pp. 2299-2306, 2016.( in Chinese) Article (CrossRef Link).

[13] Suarez-Tangil, G., S.K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L.Cavallaro, "DroidSieve: Fast and accurate classification of obfuscated android malware," in *Proc. of the Seventh ACM on Conference on Data and Application Security and Privacy*, *ACM*, pp. 309-320, 2017. Article (CrossRef Link).

[14] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for Android malware detection," in *Proc. of Computers and Communication (ISCC), 2015 IEEE Symposium on, IEEE*, pp. 714-720, 2015. Article (CrossRef Link).

[15] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *ACM*, pp. 1507-1515, 2017. Article (CrossRef Link).

[16] Y. Luan, S. Watanabe, and B. Harsham, "Efficient learning for spoken language understanding tasks with word embedding based pre-training," in *Proc. of Sixteenth Annual Conference of the International Speech Communication Association*, pp. 1398-1402, 2015. Article (CrossRef Link).
[17] E. Raff, J. Sylvester, and C. Nicholas, " Learning the PE Header, Malware Detection with Minimal Domain Knowledge," in *Proc. of the 10th ACM Workshop on Artificial Intelligence and Security, ACM*, pp. 121-132, 2017. Article (CrossRef Link).
[18] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Android Malware Detection using Deep Learning on API Method Sequences," 2017. Article (CrossRef Link).
[19] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," in *Proc. of 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), IEEE*, pp. 104-111, 2016. Article (CrossRef Link).
[20] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, pp. 114-123, 2016. Article (CrossRef Link).
[21] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to android malware feature learning and detection," in *Proc. of Trustcom/BigDataSE/I SPA, 2016 IEEE, IEEE*, pp. 244-251, 2016. Article (CrossRef Link).
[22] VirusTotal. Article (CrossRef Link).
[23] Contagio Mobile Malware Mini Dump. Article (CrossRef Link).
[24] APKPure. Article (CrossRef Link).
[25] 360 market. Article (CrossRef Link).
[26] Apktool. Article (CrossRef Link).
[27] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Proc. of Advances in neural information processing systems*, pp. 431-439, 2013. Article (CrossRef Link).
[28] Q. Le, and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. of International Conference on Machine Learning*, pp. 1188-1196, 2014. Article (CrossRef Link).
[29] J. Saxe, and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. of Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on, IEEE*, pp. 11-20, 2015. Article (CrossRef Link).
[30] G. E. Hinton, S. Osindero, and Y. W. The, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527-1554, 2006. Article (CrossRef Link).
[31] deep-belief-network. Article (CrossRef Link).
[32] Y. Shao, J. Ott, Y. J. Jia, Z. Qian, and Z. M. Mao, "The misuse of android unix domain sockets and security implications," in *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM*, pp. 80-91, 2016. Article (CrossRef Link).

**Tieming Chen,** received the Ph.D. degree in software engineering from Beihang University, Beijing, China, in 2011. He is currently a professor with the College of Computer Science and Technology, Zhejiang University of Technology. His research interests include data mining and cyberspace security.

**Qingyu Mao,** received the B.S. degree in computer science & technology from Zhejiang A&F University in 2016. He is currently pursuing the M.S. degree with the School of Computer Science & Technology and Software Engineering, Zhejiang University of Technology. His research interests include data mining and mobile security.

**Mingqi Lv,** received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2012. He is currently an assistant professor with the College of Computer Science and Technology, Zhejiang University of Technology. His research interests include ubiquitous computing, data mining and human computer interaction.

**Hongbing Cheng,** associate professor, was born in 1979. He is a post doctor candidate in State Key Laboratory for Novel Software Technology at Nanjing University. His research interest includes crptography and information security, computer communications and networks, cloud computing.

**Yinglong Li,** received the PhD degree computer science from Renmin University of China, Beijing, China, in 2014. Currently, he is a lecturer of computer science in school of computer science and technology, Zhejiang University of Technology. His research interests include edge-computing and privacy protection in Internet of Things (IoT).