ICIC Express Letters
Part B: Applications
Volume **7**, Number **7**, July **2016**

 ISSN 2185-2766
pp. 1505–1510

APPLICATION OF DEEP BELIEF NETWORKS FOR ANDROID MALWARE DETECTION

Jia Yang[1], Huixiang Zhang[1], Baolei Mao[1] and Chunlei Chen[2]

[1]School of Automation
Northwestern Polytechnical University
No. 127, West Youyi Road, Xi'an 710072, P. R. China
zhanghuixiang@nwpu.edu.cn

[2]School of Computer Engineering
Weifang University
No. 5147, Dongfeng Street, Weifang 261061, P. R. China

Received December 2015; accepted March 2016

Abstract. *Due to the openness of Android OS, malicious applications grow rapidly, which makes the Android malware detection more and more urgent. In this paper, efforts are made to apply the deep belief network to Android malware detection. A deep belief network (DBN) model combined with a back propagation (BP) neural network classifier is constructed to detect Android malware. A DBN-BP model is trained, and effects of different parameters are examined by experiments. The performance of the DBN-BP classifier outperforms other four commonly used machine learning algorithms, including BP, KNN, SVM and Native Bayes. By training the DBN-BP model with 700 API features, TPR reaches 96%, TNR rises to 99.8%, and F-score exceeds 96.2%. The DBN-BP classifier is an effective choice when high dimensional Android API feature sets are used to detect malicious applications.*
Keywords: Android, Malware detection, Deep learning, Deep belief network, API

1. **Introduction.** Android has become the most popular mobile operating system. Just as reported by the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, Android dominated the market with an 82.8% share in 2015 Q2 [1]. However, Android malware is also growing rapidly. In 2014 Q1, 275 new Android malware families have been found according to the mobile threat report by F-Secure [2]. Consequently, it is urgently required to develop effective Android malware detection mechanism.

Android operation system employs a permission-based security model. It is natural for researchers to detect Android malware based on the permissions requested by applications [3]. However, the requested permissions declared in the manifest file may not be the actual permissions required by an application. The work of [4] illustrated that approximately one-third of evaluated applications were over privileged. Google does not provide a complete mapping to the permissions that the API call may need, and it is not trivial work to identify the actually required permissions [5]. Therefore, it is better to directly use API as the classification features, rather than try obtaining the required permissions.

However, there are a large number of APIs used in Android applications. The API-based classifiers are usually trained with high dimensional feature vectors. Existing works are mainly based on the shallow architecture [3,6], such as support vector machine (SVM) and back propagation (BP) neural network. These shallow learning algorithms' learning ability is limited, in terms of complex eigenvector. They are not suitable for high dimensional classification. As a new research topic of machine learning, deep learning adopts the multilayer neural network training, and is a promising solution to the aforementioned problems [7-9]. The deep learning mechanism has capacity for self-taught learning features through

greedy layer-wise training method, and it has significantly greater representational power than shallow learning.

In this paper, a deep belief network (DBN) model combined with a back propagation (BP) neural network classifier is constructed to detect Android malware. By extracting the API calls of applications as classification feature instead of permissions, the classifier can distinguish malicious and benign sample more effectively. By adopting the DBN model, the classifier performs well even with high dimensional API features. The remainder of the paper is organized as follows. In Section 2, the overall framework for Android malware detection is introduced; the DBN and its training process are depicted in Section 3; then we discuss how to choose the parameters of DBN and compare its performance with different classifiers in Section 4. Finally, we conclude the paper in Section 5.

2. **Approach Overview.** Figure 1 illustrates the architecture of the overall framework for Android malware detection.

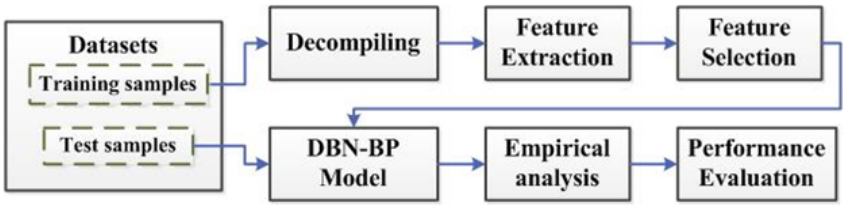

FIGURE 1. The overall framework for Android malware detection

A total of 25255 Android applications are collected in our work. There are 22476 benign samples downloaded from various Android markets. Moreover, there are 2779 malicious samples collected from Android Malware Genome Project [10] and DroidAnalytics Project [11]. 2000 benign samples and 200 malicious samples are randomly chosen as the test set, and the rest are used to train the classifier.

Apktool [12] is used to disassemble the Android application's classes.dex file into a directory tree of smali files. There are more than 200 million API records extracted from those smali files. The APIs which are not declared in the Android API level 21 documents are removed. After that, there are 39303 Android APIs left. The difference value of invoked ratio is used to select the feature API [6]. For a single Android API, the difference value of invoked ratio is denoted as R_d, and it can be calculated as follows.

$$R_b = n_b/N_b \tag{1}$$

$$R_m = n_m/N_m \tag{2}$$

$$R_d = R_m - R_b \tag{3}$$

Here, N_b and N_m are the total numbers of benign and malicious samples, respectively. n_b and n_m are the invoke counts of a single Android API in benign and malicious samples, respectively. The API is selected if its invoked ratio of malware samples is more than that of benign samples, i.e., $R_d > 0$. There are 8616 Android APIs that match the condition.

A DBN-BP classifier is constructed to detect Android malware. DBN is adopted for high dimensional feature extraction, and its output feature vector is taken as the initial value of the BP classifier. The DBN-BP classifier is fine-tuned with different parameters to achieve favorable performance. The performance of the DBN-BP classifier is compared with four commonly used machine learning algorithms, including BP, KNN, SVM and Native Bayes.

The following metrics are adopted to measure the classification performance:

1) True positive ratio (TPR): the ratio of the number of malware samples correctly classified over the total number of malware samples.

$$TPR = TP/(TP + FN) \tag{4}$$

2) True negative ratio (TNR): the ratio of the number of benign samples correctly classified over the total number of benign samples.

$$TNR = TN/(FP + TN) \tag{5}$$

3) *F-score*: the harmonic mean of Recall and Precision.

$$F\text{-}score = 2*TP/(2*TP + FP + FN) \tag{6}$$

Here TP is the number of malware samples correctly classified. FN is the number of malware samples misclassified as benign applications. FP is the number of benign samples incorrectly detected as malware. TN is the number of benign samples correctly classified.

3. **Malware Detection Based on DBN-BP Model.** The deep belief network (DBN) is a type of deep neural network, and consists of multiple layers of restricted Boltzmann machines (RBMs). Each layer in a DBN acts as a feature detector on its inputs. From this perspective, the DBN is trained in an unsupervised way [7]. Therefore, a BP neural network classifier is appended at the end of the DBN to classify the benign and malware applications.

To train the DBN-BP model, the binary feature vectors are built. If the APK sample contains a selected API feature, the corresponding bit in the feature vector is set to 1; otherwise is set to 0. At last, a class label is appended at the end of a feature vector to show that the vector belongs to a benign or a malicious application. These binary feature vectors are typically sparse. The DBN algorithm has a good advantage for recognition of sparse feature vector. Compared to the other deep learning algorithms, the DBN structure is easier to implement layer-wise learning. It uses smaller space, and handles a large amount of data faster than the others [7,8].

The training of DBN-BP model can be divided into two steps. In the first step, the binary vectors of Android APIs are taken as the input of DBN. Within the DBN, each hidden RBM layer serves as the visible layer for the next. With N layers trained in this unsupervised greedy manner, a meaningful representation of data can be extracted. In the second step, a BP classifier is appended to the last RBM layer. The output of the BP classifier is binary. Here, 1 is used for malware application, and 0 for benign application. The classification result is compared to the training sample's class label. The error is then back propagated to the DBN. Because of the effective feature reduction and representation

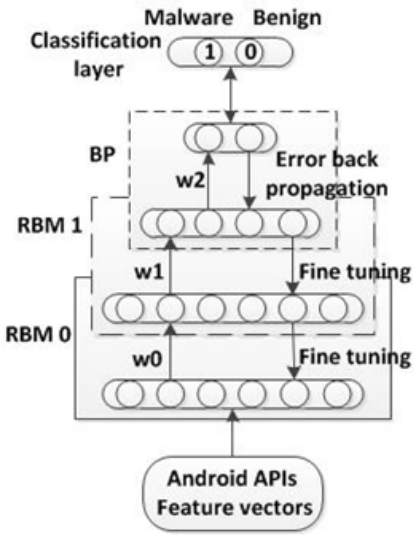

FIGURE 2. A typical two-layer of DBN-BP structure

by multiple RBM layers, the BP classifier can fast converge to an optimal result. Figure 2 shows a typical two-layer of DBN-BP structure.

4. Experiment Result and Discussion.

4.1. The training of DBN-BP model.

As is discussed in the previous section, there are several factors which affect the performance of DBN-BP model, including the number of hidden RBM layers, the number of neurons in each RBM layer, and the training steps. Therefore, we investigated on the influence of the three parameters through experiments.

Experiment 1: Altering the number of hidden RBM layers. In the experiment, a feature set of 300 Android APIs is constructed by setting the difference value of invoked ratio to 0.137. The number of hidden RBM layers rises from one to four layers, and there are 150 neuron nodes in each layer. The results are illustrated in Table 1. The metric of TPR reaches its maximum by using a DBN-BP model with two hidden RBM layers. However, if the number of RBM layers continues growing, the metric of TPR falls to 0.88 when a four-layer DBN-BP model is used. The changed trend of TNR and *F-score* are basically the same as TPR. More complex DBN structures are not superior to a two-layer structure in this experiment. The reverse error may accumulate with the increasing of hidden layers, which causes the falling of the performance. Therefore, a two-layer DBN-BP model is good enough for Android malware detection in this experiment.

Table 1. The performance metrics when altering the number of hidden RBM layers

Layers	TPR	TNR	*F-score*
1	0.915	0.937	0.919
2	**0.925**	**0.995**	**0.934**
3	0.915	0.996	0.936
4	0.880	0.992	0.900

Experiment 2: Tuning the number of neurons in a layer. The above feature set is used in the experiment. A one-layer DBN combined with BP classifier is built to detect malware. The number of neuron nodes in the one-layer DBN is varied from 300 to 100. The results are shown in Table 2. We found that 300 nodes in the layer was the best choice. If decreasing the number of node, the metric of TPR falls from 0.92 to 0.85, and the other two metrics also changed in downward trend. This is because it leads to a loss of information using less neuron nodes in the one-layer DBN. Therefore, the number of nodes in the first layer should be set close to the dimension of the input feature vector. This would let the DBN model learn representative features.

It is worth noting that the metric of TPR grows in discordance with overall trend when the number of nodes is 150. However, the metrics of TNR and *F-score* are in a significant decline. This is attributed to a local convergence which leads to a small amplitude in the training process. The overall performance shows a declining trend.

Table 2. The performance metrics when tuning the number neurons in a layer

Nodes	TPR	TNR	*F-score*
[300]	**0.920**	**0.995**	**0.934**
[250]	0.900	0.995	0.927
[200]	0.885	0.995	0.921
[150]	0.915	0.937	0.919
[100]	0.850	0.990	0.871

TABLE 3. The performance metrics when adjusting the number of training steps

Steps	TPR	TNR	$F\text{-}score$	Steps	TPR	TNR	$F\text{-}score$
1	0.900	0.995	0.923	20	0.920	0.997	0.943
5	0.925	0.992	0.923	25	0.905	0.993	0.919
10	0.930	0.987	0.902	30	0.885	0.992	0.903
15	**0.925**	**0.997**	**0.948**				

Experiment 3: Adjusting the number of training steps. A one-layer DBN combined with BP classifier is used in the experiment. The number of neuron nodes in the one-layer DBN is set to 290. The number of training steps varied from 1 to 30. The results are shown in Table 3. With the increment of training steps, the performance of the DBN-BP model gets better. When the number of steps increases to 15, the metric of TPR, TNR and $F\text{-}score$ reach their maximum, i.e., 0.925, 0.997 and 0.948 respectively. After that, the performance declaims with the increment of steps. Excessive steps lead to over-training of the DBN-BP model, which makes the model fit the noise in the training data and select unrepresentative characteristics in the samples.

4.2. **Performance comparison.** In this subsection, the classification performance of the DBN-BP model is compared with four other typical machine learning models, including Naive Bayes, BP, KNN and SVM. According to the difference value of invoked ratio, four feature sets of Android APIs are chosen for the comparison. The length of these API feature sets are varied in $[100, 300, 500, 700]$. Moreover, there are 147 kinds of official permissions declared in Android SDK [6]. These permissions are selected as a feature set to train the five models. A one-layer DBN combined with BP classifier is used for the comparison. The results are demonstrated in Figure 3.

By using the Android API feature sets, the classification is higher than that of using the permission set. Therefore, the Android APIs are preferred to detect the malware. During the comparison, the DBN-BP classifier almost outperforms the others. Meanwhile, the

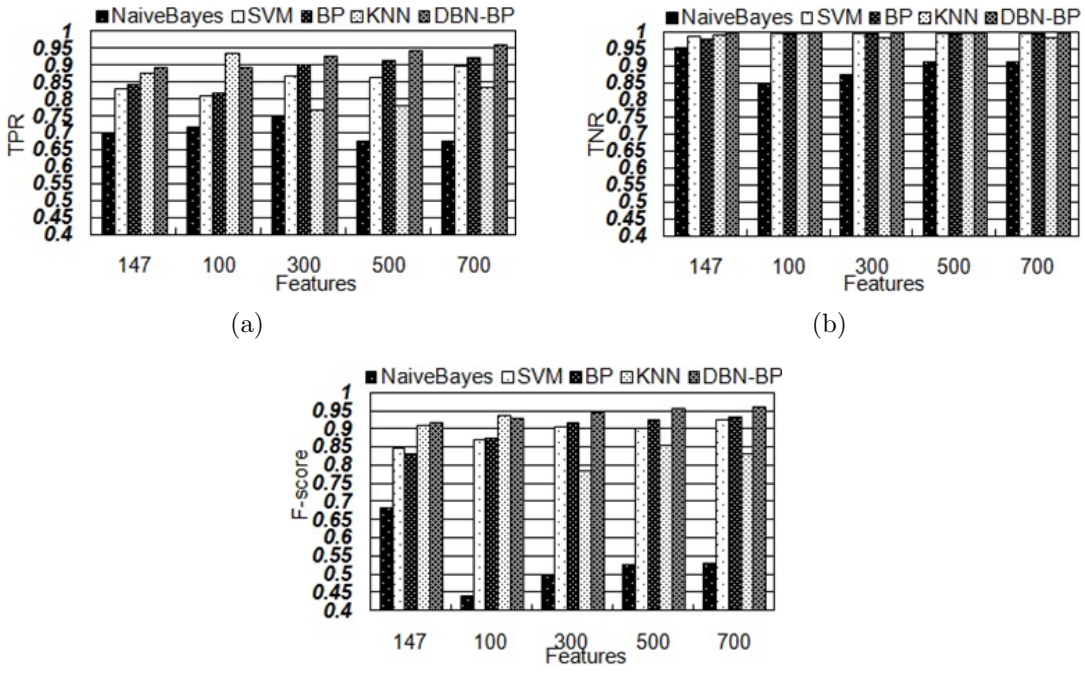

FIGURE 3. The performance evaluation with the five feature sets

performance of the DBN-BP model grows with the number of Android APIs. By training the DBN-BP model with the 700 API features, the TPR reaches 96%, the TNR rises to 99.8%, and *F-score* exceeds 96.2%. It indicates that the DBN-BP classifier has a better performance when the input feature vectors are of very high dimension. Although the KNN classifier works better than the DBN-BP classifier on the 100 API feature set, the performance drops off on high dimension feature sets.

5. **Conclusions.** In this paper, the application of the deep belief network to Android malware detection is studied. We found that a one layer or two layers DBN-BP model is adequate for Android malware detection, and the number of nodes in the first layer should be set close to the dimension of the input feature vector. We should choose proper training steps to avoid over-training of the DBN-BP model. Compared to the other four machine learning classifiers, DBN-BP exhibits a very close performance and even outperforms the other classifiers when the input feature vectors are of very high dimension. Therefore, the DBN-BP is an effective model when using high dimensional Android API feature sets. However, the parameters of DBN-BP model are manually fine-tuned through extensive experiments. In the future work, we will be dedicated to developing a self-adaptive model.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China (No. 61303224) and the Graduate Starting Seed Fund of Northwestern Polytechnical University (No. Z2015126), and the Science and Technology Development Program of Weifang (No. 2015GX008).

REFERENCES

[1] IDC, *Smartphone OS Market Share*, http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015.
[2] F-Secure, *Mobile Threat Report*, https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014_print.pdf., 2014.
[3] B. Sanz, I. Santos, C. Laorden, X. U. Pedrero, J. Nieves, P. G. Bringas and G. A. Marañón, MAMA: Manifest analysis for malware detection in Android, *Cybernetics and Systems: An International Journal*, vol.44, nos.6-7, pp.469-488, 2013.
[4] A. P. Felt, E. Chin, S. Hanna, D. Song and D. Wagner, Android permissions demystified, *Proc. of the 18th ACM Conference on Computer and Communications Security*, Chicago, IL, USA, pp.627-638, 2011.
[5] K. W. Y. Au, Y. F. Zhou, Z. Huang and D. Lie, PScout: Analyzing the Android permission specification, *Proc. of the 19th ACM Conference on Computer and Communications Security*, Raleigh, NC, USA, pp.217-228, 2012.
[6] H. Zhang, C. Chen, J. Li and Y. Luo, Study on minimal feature selection for Android malware detection, *The 10th International Conference on Innovative Computing, Information and Control*, Dalian, China, pp.1-6, 2015.
[7] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science*, vol.313, no.5789, pp.504-507, 2006.
[8] Y. Bengio, Learning deep architectures for AI, *Foundations and Trends in Machine Learning*, vol.2, no.22, pp.1-127, 2009.
[9] S. Jürgen, Deep learning in neural networks: An overview, *Neural Networks*, vol.61, pp.85-117, 2015.
[10] Y. Zhou and X. Jiang, Dissecting Android malware: Characterization and evolution, *IEEE Symposium on Security and Privacy (SP)*, vol.95, no.109, pp.20-23, 2012.
[11] M. Zheng, M. Sun and J. C. S. Lui, DroidAnalytics: A signature based analytic system to collect, extract, analyze and associate Android malware, *Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp.163-171, 2013.
[12] Apktool, *A tool for Reengineering Android APK Files*, http://code.google.com/p/Android-apktool/.