Hindawi
Security and Communication Networks
Volume 2021, Article ID 9994588, 10 pages
https://doi.org/10.1155/2021/9994588

Research Article

A Hybrid Modeling of Mobile App Dynamics on Serial Causality for Malware Detection

Meilin Liu,[1] **Songjie Wei**,[2] **and Pengfei Jiang**[2]

[1]*Shanghai Jiao Tong University, Shanghai 200240, China*
[2]*Nanjing University of Science and Technology, Nanjing 210094, China*

Correspondence should be addressed to Meilin Liu; meilin.liu@sjtu.edu.cn

Received 11 March 2021; Revised 31 August 2021; Accepted 25 September 2021; Published 15 October 2021

Academic Editor: Leandros Maglaras

The popularity of smart phones has brought significant convenience to people's lives, but also there are many security problems. In recent years, malicious applications are increasingly rampant, which threaten users and society as security challenges to network reliability and management. However, due to neglecting the sequential features between network flows, existing malicious application recognition methods based on network traffic analysis have low recognition accuracy. Based on the network traffic characteristics of Android applications, this paper firstly applies Long Short-Term Memory network-based variational Auto-Encoder to extract the sequential feature of the application running time. Then, we design the BP neural network for initial classification and connect the class vector output of the BP neural network with the original data. The output is fed into the cascade forest for further feature learning and classification. The integrated methods are easy to implement with data independency and efficiency. We conduct experiments to evaluate the proposed with Android malware dataset CICAndMal2017, with a 97.29% high accuracy, comparatively significant precision and recall rates when benchmarked against other methods.

1. Introduction

In recent years, with the development of the mobile Internet, the Android platform has won a monopolistic share in the global market by virtue of flexibility, open source development processes, and inexpensive hardware products. At the same time, Android has maintained a continuous growth trend. However, the openness and popularity of the Android platform have also attracted a variety of malicious attackers. These attacks can take the form of system attacks, data theft, and resource misuse. These malicious behaviors seriously endanger the security and privacy of Android users. Therefore, identifying and detecting malicious applications on the Android platform has become an urgent problem for security researchers.

Based on the successful experience of malicious application detection on the PC side, traditional mobile malicious application detection products are mostly based on the mobile terminal to achieve malicious application detection, mainly including static detection and dynamic system call detection methods. Using static signatures to detect malicious code is the most common and direct method in mobile terminal. Almost all antivirus software in the industry is based on this method. However, the method based on static feature detection has inevitable hysteresis. It needs to update the malicious rule base frequently to maintain the ability to recognize the latest threats. It has insufficient ability to discover unknown malicious applications. At the same time, it is necessary to deploy malicious application detection software on the intelligent terminal to achieve discovery. Blocking malicious applications from installation or execution and implementing terminal scanning consume considerable system resources [1]. In addition, with the development of malicious code diversity and code obfuscation techniques, this static detection method faces increasing challenges. For the system call dynamic feature detection method, due to the complexity of the system call behavior and the difficulty of the detection technology, it is difficult to directly deploy the malicious terminal to implement malicious application identification.

When a mobile smart terminal performs a malicious action, the malware shows different characteristics in network interaction. Research [2] finds that more than 90% of malware-infected mobile terminals are controlled by botnets and control malware behavior through network or SMS instructions. This discovery provides the possibility to explore malware detection methods based on network behavior characteristics. Malicious application detection based on network behavior is intended to detect malicious application behavior from the perspective of network traffic data, which has been paid close attention by academia and industry. The biggest advantage of this method is that it analyzes the traffic data generated by the user online and does not consume the user's device resources. The malware detection model based on traffic analysis can be deployed to network access points or to the cloud. In addition, this method is easier to be implemented than dynamic analysis.

The malicious application identification method based on the statistical characteristics of the network flow has the advantages of low computation cost, no need to inspect the contents of the data packet, and the ability to be used for unencrypted and encrypted traffic while protecting privacy. Although the statistical information of network traffic can effectively describe the behavior characteristics of network traffic to some extent, they lose the dynamic characteristics of traffic behavior changing with time. When the network scales or the network application environment changes, these statistics may also change accordingly, so it is unable to accurately and reliably describe the behavior of network application traffic.

To sum up, based on the statistical characteristics of the network flow of Android applications, this paper studies the temporal relationship between network flows, proposes an Android malware detection method, and verifies the method on the open Android malware data set CICAndMal2017. The main contributions of this paper are as follows:

(i) We integrate Long Short-Term Memory network (LSTM) with the variational Auto-Encoder and propose a new method using the LSTM-based variational Auto-Encoder (LSTM-VAE) to extract the sequential characteristics of the data. Multiple sequential data chunks are reconstructed into one piece, which not only reduces the amount of data, but also reconstructs the data presentation to reflect the sequential relationship of the original data.

(ii) We propose a powerful classifier for representing learning ability by combining a BP neural network with a cascade forest. The reconstructed data is first fed into the BP neural network, where the output is a class vector. Then, the combination of the class vector and the reconstructed data is introduced into the cascade forest, and finally the discrimination result is obtained. Experimental results show that the procedure has significant classification effectiveness and efficiency.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 introduces the detection process and the detection model used (LSTM-VAE and cascade forest). Section 4 presents the experimental results and analysis. Finally, Section 5 provides conclusions and future work.

2. Related Work

Current classification technology for Android applications is mainly based on the static code features and dynamic running characteristics of the applications. Static feature classification is generally obtained by decompiling the application permissions [3], API call list [4], and other features for classification. Idrees and Rajarajan [5] propose a method for detecting malware by analyzing the permissions and intent combination features of Android applications, which is further combined with machine learning algorithms to classify applications. Wu et al. [6] give the DroidMat detection tool, extracting several static features from the AndroidManifest.xml file as the characteristics of Android software and using the classification algorithm to analyze the Android software to be detected. Dhaya and Poongodi [7] design a malware detection system based on the N-gram algorithm. In this system, the behavior characteristics of software are first obtained through static analysis. Then, the N-gram algorithm is used for classification and identification, so that the category of the software is either benign or malicious. Static analysis is even extended to handle obfuscated Android Apps for malware detection, such as in DroidSieve [8]. However, the classification method based on static features is incapable of resisting advanced stealth techniques such as code confusion, dynamic update, and real-time loading commonly used by malware authors [9].

The dynamic feature classification is used to obtain the running characteristics or behavior characteristics of the application for pattern specification. Operating characteristics include battery temperature and power consumption, and memory and CPU load status. Approaches such as DroidScribe [10] apply various models to generate high-accuracy App prediction based on the runtime dynamics with limited information. More complex behavioral features utilize feature description methods such as API call sequence [11] and network behavior [12]. API call based App execution dynamics have been proved effective in unveiling malware behaviors [13]. Bläsing et al. [14] propose a dynamic detection tool called AASandbox, which executes Android software in a completely isolated environment and detects Android malware by analyzing the low-level interactive logs obtained during execution. Wang et al. [15] introduced a method for detecting malicious advertisements called AdHoneyDroid, building a crawler to collect applications on the Android market and manually collected the ad library and its vulnerabilities. The API sandbox and TaintDroid are used to detect attack events and to store malicious advertisements in a database for future analysis. Dynamic analysis technology can effectively resist the polymorphism and code obfuscation techniques of malicious code.

Machine learning-based detection methods mainly include support vector machines [16], random forests [17], and decision trees [18]. Narudin et al. [19] propose a solution for malware detection that uses machine learning to evaluate various network traffic characteristics. Shabtai et al. [20] construct a new framework for Android malware detection, first monitoring the features obtained from Android devices,

and then using machine learning methods to train to determine whether the software is malware. However, these methods ignore the timing relationship between data, so the classification accuracy is not stably high. We believe machine learning methods should be better constructed and integrated to reveal the dynamic patterns in malware behaviors, from both the inside code characteristics and the outside behavioral observations.

For all the machine-based detection methods, the detection models are derived from some training dataset. The models may be restricted by the training set statics [21]. Although malware may vary in code appearance, but their intrinsic operation procedure should not change too much as long as the same kind of malicious behaviors are carried out. Researchers are looking for such maintained-over-evolution App characteristics, such as API calls [22], and data access pattern [23].

3. Detection Mechanism

3.1. Overview. The proposed method takes the full advantage of the sequential characteristics in the application data and automatically extracts key features to evaluate and differentiate these applications. The process operates as in Figure 1.

(a) Preprocessing the feature data of Android applications, including eliminating nonsignificant features and data normalization

(b) Extracting the sequential features of the application data using LSTM-VAE

(c) Applying BP neural network for initial classification, where the classification result is aligned with the extracted data for improved classifier performance

(d) Input the above data into the cascade forest and finally get the detection results

3.2. Sequence Feature Extraction Model. VAE [24] is a variant of Auto-Encoder (AE) and is an unsupervised learning method. VAE consists of a sequentially connected network of encoders and decoders. It sets the target of the decoder equal to the input of the encoder. The encoder network learns the compressed representation of the input, and the decoder network reconstructs the target from the compressed representation. Encoder and decoder can be neural network models such as BP neural network, convolution neural network, and recurrent neural network. The difference between input and reconstruction input is the reconstruction error. During the training period, the VAE minimizes the reconstruction error as the objective function.

The basic idea of applying the VAE algorithm is that there is a hidden variable z_k instead for each input application data point $x_k \in u_i$. Therefore, the final output $\hat{x}_k$ can be generated by a certain probability distribution $p\theta(x_k|z)$, which is assumed to be a Gaussian distribution according to common sense herein. The decoder function $f_\theta(z_k)$ is capable of generating the final parameters of the generated distribution and is itself constrained by a uniform set of parameters θ. At the same time, the encoder function $g\theta(x_k)$ is able to generate a parameter for each probability distribution $q_\varnothing(z_k|x_k)$, which is also constrained by the parameter θ.

The variational derivation indicates that an approximate distribution $p(z_k)$ needs to be found instead of the distribution $q_\varnothing(z_k|x_k)$. On the one hand, the loss function of VAE is the reconstruction error of data, which is measured by mean square error. On the other hand, the difference between the distribution of latent variables and the distribution of unit Gauss is usually measured by a function called Kullback–Leibler (KL) divergence. Therefore, the lower bound objective function for a single input data point is

$$L(\theta, \varnothing; x_k) = -D_{KL}\left[q_\varnothing(z_k|x_k)\middle\|p(z_k)\right] + E_q[\log\ p_\theta(x_k|z_k)]. \tag{1}$$

VAE usually uses BP neural network as encoder and decoder, but BP neural network does not have a memory function. A Long Short-Term Memory Network [25] is able to take the advantage of historical information and selectively remember important historical information. Typical LSTM consists of an input gate, a forgetting gate, and an output gate. The function of the forgetting gate is to let the LSTM forget the information that was not used before. The function of the input gate is to supplement the latest memory from the current input. The function of the output gate is to determine the output of the information. LSTM is very suitable for modeling time series with long time sequential correlation. The LSTM cell structure is shown in Figure 2.

Network traffic is essentially a kind of sequential data, which is a one-dimensional byte stream organized by hierarchical structure. Among them, bytes, packets, sessions, and traffic are analogous to characters, words, sentences, and articles in the field of natural language processing. In addition, by analyzing and extracting some characteristics of network traffic, we can get many attributes, such as flow duration, message arrival time interval, and message length. Although these attributes are independent, they are related on the whole, and they reflect the nature of the traffic together. We combine LSTM and VAE to propose a Long Short-Term Memory network-based variational Auto-Encoder. We use LSTM as an encoder and decoder in VAE, which introduces the time dependence of time series data into VAE. LSTM-VAE can model time series data and extract the timing relationship of sequence data. Although the forgetting gate structure in LSTM is designed to reduce the problem of gradient explosion to a certain extent, if the input sequence is long enough, the LSTM structure may still suffer from the problem of gradient explosion. Under such problem, the neural network model cannot be updated from the training data, the loss function may change significantly during the updating process, and even the loss value cannot be obtained. Therefore, we use gradient clipping to prevent the occurrence of gradient explosion. Gradient clipping means checking and limiting the gradient during the training time. The LSTM-VAE model designed is shown in Figure 3.

In the above design, μ represents the mean, σ represents the logarithm of the variance, and both μ and σ are sampled from the standard positive distribution using the fully

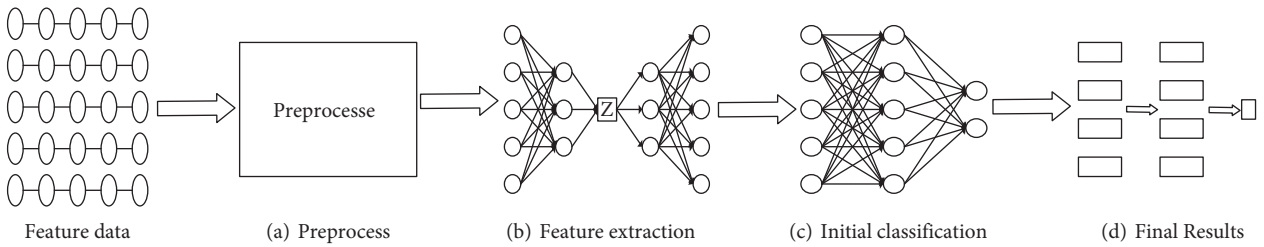

Figure 1: Application detection process.

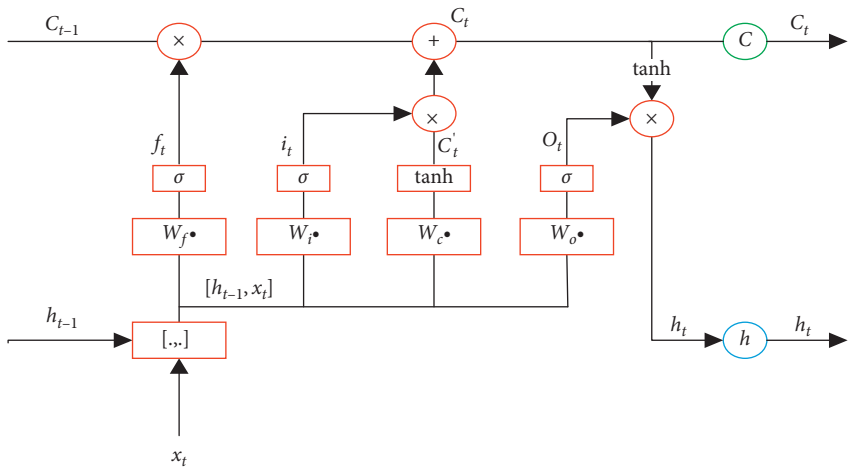

Figure 2: The structure of LSTM cell.

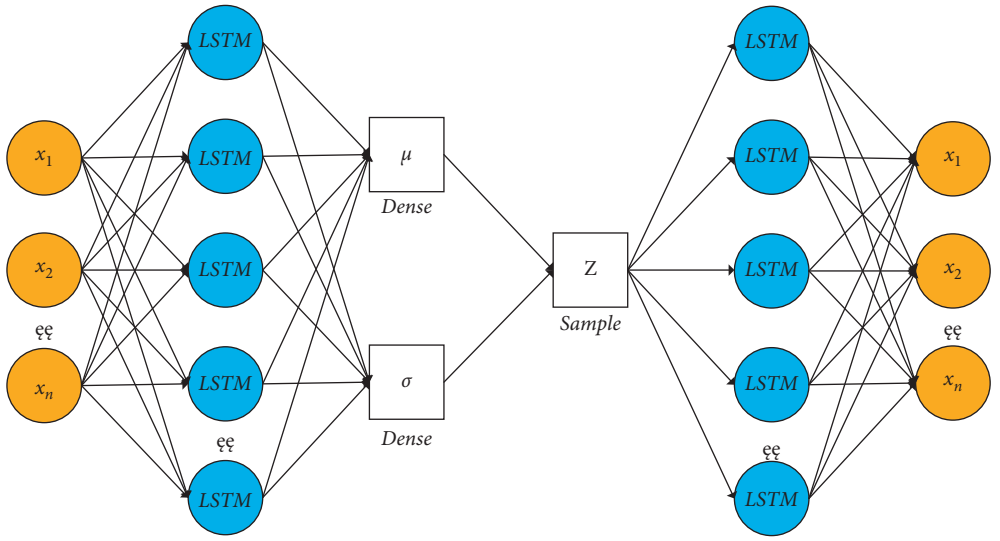

Figure 3: The structure of LSTM-VAE model.

connected layer. Both the encoder and the decoder consist of an LSTM layer with 50 nodes. The parameter initialization method uses the Xavier initialization method. The dimension size of the Z (latent space) is the same as the dimension of the input data. Z (sampled data) is the characteristic data we need. The optimizer uses the Adam method. The data we need is sampled data. We reconstruct 10 pieces of data from each application into one piece of data with the same dimension, which reduces the amount of data to one-tenth of the original amount and extracts the most significant characteristics of the time series representing the application dynamics.

3.3. Classification and Detection Model. Deep Forest [26] is an ensemble learning method based on the decision tree. It has strong representation learning ability and consists of multigrained scanning and a cascade forest. It is also called a

multigrained cascade forest. Multigrained scanning is used to extract temporal or spatial features, and the forests are cascaded through multiple forests to obtain classification results. Here, we only use the cascade forest part to hand over the timing feature extraction to LSTM-VAE. The cascade forest of the deep forest uses a layer-by-layer structure similar to the deep neural network. Each layer generates a class probability vector with the length of C (number of categories). If there are N classifiers in the first layer, the C elements generated by each classifier are stitched together to form $C \times N$ element vectors, and then the source input eigenvectors are stitched together to form the input to the next layer.

Each layer of the cascade forest can be a common machine learning classifier, such as Random Forest, extra random forest, and XGBoost. However, it should be noted that the classifiers in each layer cannot be in the same type. For example, one layer is composed of random forests because the diverse structure is very important for ensemble learning [27]. Finally, each classifier outputs a class probability vector, which is averaged. The largest class is the final prediction result.

After extending a new level, the performance of the whole cascade is estimated on the verification set. If there is no significant performance gain, the training process should be terminated. Therefore, the number of cascaded intermediates is automatically determined. The cascade forest is able to determine the complexity of its model by terminating training appropriately. It makes the cascade forest suitable for training data of different sizes, not limited to large-scale training data.

In view of the excellent self-learning and self-adapting ability of BP neural network, we combine BP neural network with a cascade forest to form a powerful classifier, which can represent learning ability. We input the application data into the BP neural network. After training, the BP neural network outputs the class vector and then connects the class vector and data into the cascade forest. Through this method, we have greatly enhanced the learning ability of the cascade forest.

The BP neural network used in the experiment consists of input layer, hidden layer, and output layer. The input layer contains 68 nodes, with no activation function used. The specific structure of the hidden layer is determined in the following experiments. The activation function of the hidden layer uses the ReLu function. The output layer contains two neuron nodes, using the Softmax function, and the output value represents the discrimination probability. The optimizer uses the Adam algorithm. The loss function uses cross-entropy loss function.

Each layer of the cascade forest used in this paper has four classifiers, namely, XGBoost [28], Random Forest, ExtraTrees, and logistic regression. The maximum number of layers of the cascade forest is set to 100. If there is no improvement in accuracy within three layers of a certain cascade, the forest stops adding layers. The four classifiers are configured with parameters as shown in Table 1 below.

N_folds means using N-fold cross-validation to reduce the overfitting risk of cascade forests. *Max_depth* represents the maximum depth of the tree and is used to prevent overfitting problems. *N_estimators* represent the maximum number of weak learners. The overall structural parameters of BP neural network and cascade forest are shown in Figure 4.

Table 1: Parameters of classifiers in cascade forests.

Parameter	Classifier			
	XGBoost	Random Forest	ExtraTrees	Logistic regression
N_folds	5	5	5	5
N_estimators	20	20	20	—
Max_depth	10	None	None	—
Learning rate	0.08	—	—	—

4. Experimental Results and Analysis

4.1. Experimental Environment. All the experiments are conducted on a desktop with 8 GB of RAM, Intel Core CPU i7-7700 CPU 3.6 GHz, and Nvidia GeForce GTX 1070Ti graphics card. The programming language used is Python 3.5. The implementation of a cascade forest model is based on machine learning framework Scikit-learning (0.18.0), and the construction of LSTM-VAE and BP network is based on deep learning framework TensorFlow (1.7.0).

4.2. Dataset and Preprocessing. CICAndMal2017 [29] is a new Android malware dataset proposed by the Canadian Institute for Cybersecurity. In this approach, it runs both the malware and benign applications on real smartphones to avoid runtime behavior modification of advanced malware samples that are able to detect the emulator environment. It installed 5,000 of the collected samples (426 malware and 5,065 benign) on real devices. The dataset includes benign application samples and malicious application samples, which are divided into 4 categories including adware, ransomware, scareware, and SMS malware. Malicious samples come from 42 unique malware families. Malware usually does not express its actual malicious behavior unless it is installed and operated on a physical device. Although some datasets capture the dynamic characteristics of the samples, they only execute malware on emulators and Android virtual devices (AVDs) and avoid running applications on real smartphones.

In order to acquire a comprehensive view of malware samples, the dataset builders created a specific scenario for each malware category. They also defined three states of data capturing in order to overcome the stealthiness of advanced malware:

(i) Installation: the first state of data capturing, which occurs immediately after installing malware (1–3 min)

(ii) Before restart: the second state of data capturing, which occurs 15 min before rebooting devices

(iii) After restart: the last state of data capturing, which occurs 15 min after rebooting phones

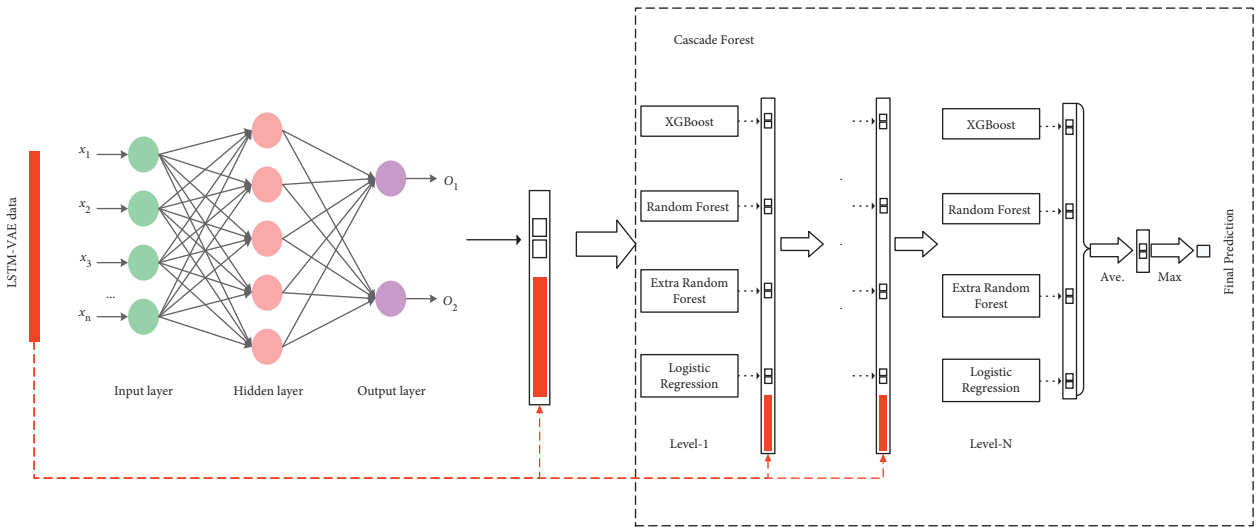

Figure 4: The overall structural parameters of BP neural network and cascade forest.

For data recording, the dataset captured the network traffic features (.pcap files) and extracted 84 features during all the three mentioned states (installation, before restart, and after restart). CICAndMal2017 is fully labeled and includes network traffic, logs, API/SYS calls, phone statistics, and memory dumps of 42 malware families.

Next, we need to preprocess the data. We sort the records for each application's data by timestamp, so that LSTM-VAE can extract sequential information. Secondly, we exclude nonnumerical features, including *Flow ID, Source IP, Source Port, Destination IP, Destination Port* and *Timestamp*. These nonnumeric features are not accepted by the neural network and are irrelevant. Thirdly, we select all the statistical characteristics of network traffic, totaling 68, some of which are shown in Table 2. Fourthly, we use 0 to fill in missing values. Finally, we use the *Z*-score method to normalize the data. The *Z*-Score method uniformly converts data of different magnitudes into the same order of magnitude and uniformly measures the calculated *Z*-Score value to ensure the comparability between the data. The *Z*-Score calculation formula is as follows:

$$x^* = \frac{x - \mu}{\delta}. \tag{2}$$

μ represents the mean of the total data, δ represents the standard deviation of the total data, and x represents the individual data. We randomly select 33% of the data as the test set and the rest for training purpose.

4.3. LSTM-VAE Feature Extraction Experiment. Due to the large amount of data, we used the small batch training method to reduce the training error. The batch size of LSTM-VAE is set to 20, and iteration is 160 times. Since the learning rate has a great influence on the convergence of the model, we set different learning rates to study the impact of learning rate on the convergence of the model. Taking an application for a benign application sample as an example, the change of the loss function at different learning rate is as shown in Figure 5.

It can be seen that when the learning rate is small, the convergence speed of the model is slow; the final convergence effect is not good, and the loss value is large. When the learning rate is higher, the convergence speed of the model is faster, and the loss function value after training is smaller. Therefore, the higher learning rate is desired.

We use LSTM-VAE to extract sequential features, reduce the data volume, and reconstruct the length of each data record the same as the original. Without GPU acceleration, the average running time of processing each application data is with a minute.

4.4. BP Network and Cascade Forest Classification Experiment. The learning rate of the BP neural network is set to 0.0001. We use the mini-batch training method, the batch size is 2000, and the total iterations are 4000 times. Experimental results show that if the number of hidden nodes in the BP network is too small, the network cannot have the necessary learning ability and information processing capability. In order to find a better BP network model, this experiment builds a BP network model with 2 to 5 layers as multilayer hidden layers. Each layer contains 80 to 120 neuron nodes. The number of neuron nodes in each hidden layer is identical. The experimental input data is the data extracted by the LSTM-VAE. During the experiment, each structure is tested 10 times, and the average classification accuracy of each BP network model is calculated. The experimental results are shown in Table 3.

It can be seen from Table 3 that as the number of layers of the hidden layer and the number of nodes per layer increase, the classification accuracy of the model is also growing. When the number of hidden layers is 4 with each layer containing 110 neurons, BP network has the highest classification accuracy. After that, the complexity of the model is increased, and the classification accuracy rate does not vary too much, indicating that the overfitting has occurred at that time. The experimental results also show that the classification effect of the neural network model does not always

Table 2: Sample selected features.

Feature	Description
Fwd_Packet_Length_Std	Standard deviation of the size of packet in forward direction
Init_Win_bytes_forward	The total number of bytes sent in the initial window in the forward direction
Init_Win_bytes_backward	The total number of bytes sent in the initial window in the backward direction
Bwd_Packet_Length_Min	Minimum of the size of packet in backward direction
Total_Length_BwdPackets	The total size of the packet in backward direction
Flow_IAT_Max	Maximum interarrival time of the packet

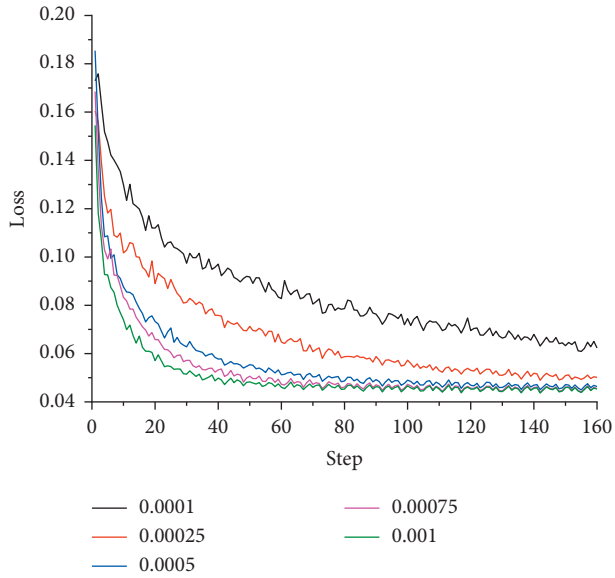

Figure 5: Loss function value of LSTM-VAE under different learning rate.

perform better with the increase of the number of layers in the hidden layer or the number of nodes in each layer. When there are too many layers and nodes in the model, the phenomenon of overfitting is more likely to occur, which performs well with training but is likely to be faint in testing. Through the above experiments and analysis, we conclude that the structure of 4 hidden layers and 110 nodes in each layer is optimal for the dataset. This will be the BP network structure used in the following experiments.

Now, we use the data from feature extraction to evaluate application classification. With GPU acceleration, the training time of BP neural network is 26 minutes, and the final classification accuracy is 90.15%. We connect the class vectors output by BP neural network with the original data for training cascade forest. In order to verify the stability of the cascade forest model, we conducted 4 experiments and added precision rate, recall rate, Area Under ROC Curve (AUC), and training time as the evaluation indexes of the model performance. The experimental results of cascade forest training using the reconstructed data link between the class vectors output from BP neural network and LSTM-VAE are shown in Table 4.

In the above four experiments, the cascade forest eventually expands four layers. It can be seen from Table 4 that the average classification accuracy of the model is 96.99%. The highest accuracy is 97.29%, and it has higher precision rate, recall rate, and AUC values. It can be seen that the performance of the cascade forest is still very stable. In the case of CPU calculation alone, the average training time is 11 minutes, and it can be observed that the training time of the cascade forest is relatively short.

Next, we experimented with preprocessed data. With GPU acceleration, the training time of BP neural network is 164 minutes, and the final classification accuracy is 80.24%. The experimental results are shown in Table 5.

In the above four experiments, the cascade forest eventually expands four layers. Because BP neural network combined with cascade forest model cannot perceive sequential features, the classification results of the model are obviously not as good as using data without LSTM-VAE to extract temporal features, and the training time of the model is also very long, but the highest classification accuracy can still reach 87.78%. It can be seen that the representation learning ability of the model proposed by the BP neural network combined with cascade forest is very powerful. It is very important to extract and utilize the sequential characteristics of applications at runtime, which is very helpful for the detection of malicious behavior.

4.5. Comparison with Other Methods. In order to verify that the proposed method is more efficient than other machine learning classification methods, we compare it with other methods, including Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), and Gradient Boosting Regression Tree (GBDT). We also benchmark with a deep learning-based detection model by using the pure LSTM for malware dynamics pattern extraction. The experimental data are the preprocessed data. The experimental results are shown in Figure 6.

As in Figure 6, in the case of only preprocessed data, the performance of other common machine learning methods is not good due to the lack of consideration of data serialization. The average classification accuracy is less than 80%, and other performance indicators are also poor.

Next, we use the data extracted by LSTM-VAE to carry out experiments. The experimental results are shown in Figure 7. When other machine learning methods use time series feature extraction data, the classification accuracy and other performance indicators are improved, but they are lower than those of our method. It can be seen that the performance of the classifier proposed in this paper is better than the common machine learning classification model.

Table 3: Classification accuracy of BP network models with different hidden layer structures.

Nodes	Layers			
	2 (%)	3 (%)	4 (%)	5 (%)
80	74.23	76.39	82.45	84.65
90	76.61	78.35	83.55	85.71
100	77.01	80.35	85.9	89.16
110	78.75	82.05	90.15	88.65
120	80.55	81.35	89.55	87.76

Table 4: Experimental results of cascade forest (LSTM-VAE data).

	Precision (%)	Recall (%)	AUC (%)	Accuracy (%)	Time (min)
1	94.63	96.07	96.03	97.29	11.6
2	93.59	95.33	95.56	96.49	10.6
3	93.78	95.51	95.73	96.89	10.8
4	94.63	96.07	96.03	97.29	10.3
Min	93.59	95.33	95.56	96.49	10.3
Max	94.63	96.07	96.03	97.29	11.6
Ave.	94.16	95.75	95.84	96.99	10.83

Table 5: Experimental results of cascade forest (raw data).

	Precision (%)	Recall (%)	AUC (%)	Accuracy (%)	Time (min)
1	83.79	87.15	86.53	87.78	32.1
2	84.23	85.78	86.45	87.32	31.7
3	83.91	88.32	85.46	86.58	33.5
4	84.76	87.49	86.01	86.29	33.1
Min	83.79	85.78	85.46	86.29	31.7
Max	84.76	88.32	86.53	87.78	33.5
Ave.	84.17	87.18	86.11	86.99	32.6

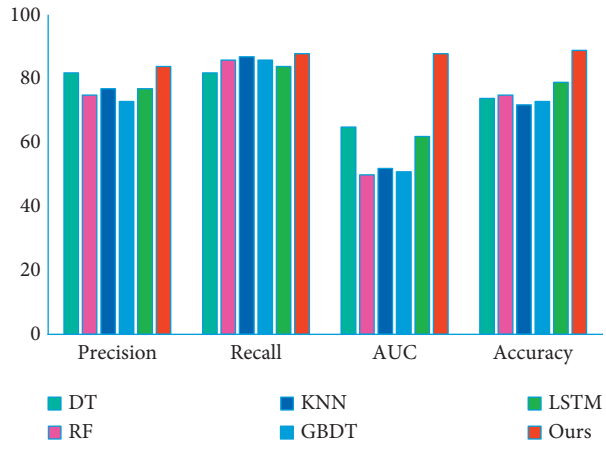

Figure 6: Comparison with other methods (raw data).

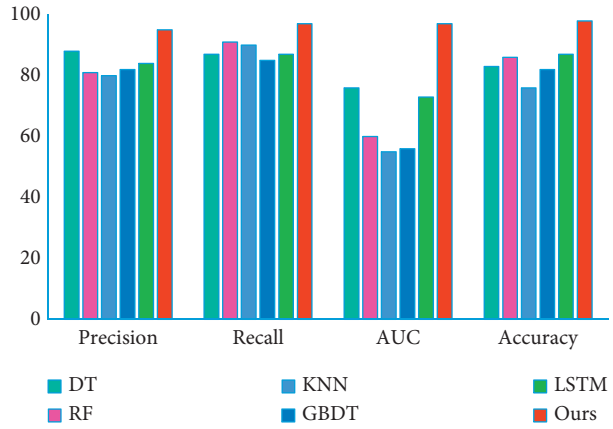

Figure 7: Comparison with other methods (LSTM-VAE data).

5. Conclusions and Future Work

Experiments show that using LSTM-VAE to extract the sequential features of network traffic of Android applications not only greatly reduces the amount of data to be processed and modeled, but also reduces the training time for the subsequent classifiers. Using the data extracted by LSTM-VAE to train the combined classifier of BP neural network and cascade forest, the average classification accuracy can be improved by at least 10% compared with those trained only with the raw data. Compared with other machine learning methods, because we extract the sequential features of network traffic dynamics and use an integration of classifiers with strong learning ability, it is superior to the common machine learning classification methods in classification performance and has a very high detection accuracy of Android malicious applications.

In future work, we plan to further summarize and extract the temporal characteristics of Android applications in network traffic dynamic. While LSTM can only use historical information, bidirectional LSTM utilizes both historical information and future information. Bidirectional LSTM greatly increases the amount of information that can be used. We may try to replace the LSTM in LSTM-VAE with bidirectional LSTM. Cascade forest can automatically adjust the depth of the model. Adaptive adjustment of the complexity of the model makes cascade forest scalable and applicable to different training datasets. We expect to apply this method to BP neural network, so that BP neural network can also automatically adjust the complexity of the model.

Learning-based malware detection is always challenged by App evolution. Our proposed method is trying to model the App dynamics reflected in network behaviors, which is not tightly bound with App intrinsic characteristics, but malicious operations. We are planning to further evaluate and prove our method for malware detection sustainability when Apps evolve, by constructing a time-lasting dataset.

Data Availability

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.

Authors' Contributions

Songjie Wei conceptualized the study; Meilin Liu was responsible for methodology; Pengfei Jiang was responsible for software.

Acknowledgments

This work was supported by SJTU research start-up fund under Grant no. AF4130058, the National Natural Science Foundation of China under Grant no. 61472189, and CERNET Innovation Project under Grant no. NGII20160105.

References

[1] P. Faruki, A. Bharmal, V. Laxmi et al., "Android security: a survey of issues, malware penetration, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[2] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2012.

[3] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.

[4] J. Park, H. Chun, and S. Jung, "API and permission-based classification system for android malware analysis," in *Proceedings of the International Conference on Information Networking (ICOIN)*, Chiang Mai, Thailand, January 2018.

[5] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *Proceedings of the IEEE International Conference on Wireless and Mobile Computing*, Larnaca, Cyprus, October 2014.

[6] D. Wu, C.-H. Mao, T. Taiwan, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: android malware detection through manifest and API calls tracing," in *Proceedings of the Asia Joint Conference on Information Security*, August 2012.

[7] R. Dhaya and M. Poongodi, "Detecting software vulnerabilities in android using static analysis," in *Proceedings of the International Conference on Advanced Communications, Control and Computing Technologies*, May 2014.

[8] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: fast and accurate classification of obfuscated android malware," in *Proceedings of the ACM Conference on Data and Application Security and Privacy*, Scottsdale Arizona USA, March 2017.

[9] H. Wang, Y. Guo, Z. Tang, G. Bai, and X. Chen, "Reevaluating android permission gaps with static and dynamic analysis," in *Proceedings of the Global Communications Conference (GLOBECOM)*, December 2015.

[10] K. Dash, "Droidscribe: classifying android malware based on runtime behavior," in *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, May 2016.

[11] Ki Youngjoon, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis,"

International Journal of Distributed Sensor Networks, vol. 11, no. 6, Article ID 659101, 2015.

[12] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, 2019.

[13] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2019.

[14] T. Bläsing, "An android application sandbox system for suspicious software detection," in *Proceedings of the International Conference on Malicious and Unwanted Software*, October 2010.

[15] D. Wang, "AdHoneyDroid--Capture malicious android advertisements," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, AZ, USA, November 2014.

[16] P. O'Kane, S. Sezer, K. McLaughlin, and E. G. Im, "SVM training phase reduction using dataset feature filtering for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 500–509, 2013.

[17] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, no. 8, pp. 206303–206324, 2020.

[18] I. Delibalta, K. Gokcesu, M. Simsek, L. Baruh, and S. S. Kozat, "Online anomaly detection with nested trees," *IEEE Signal Processing Letters*, vol. 23, no. 12, pp. 1867–1871, 2016.

[19] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[20] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[21] K. Ron and D. Hendler, "DAEMON: dataset/platform-agnostic explainable malware classification using multi-stage feature mining," *IEEE Access*, no. 9, pp. 78382–78399, 2021.

[22] X. Zhang, "Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event USA, November 2020.

[23] H. Cai, "Assessing and improving malware detection sustainability through App evolution studies," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 2, pp. 1–28, 2020.

[24] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[25] T. Ergen, A. H. Mirza, and S. S. Kozat, "Unsupervised and semi-supervised anomaly detection with LSTM neural networks," 2017, https://arxiv.org/abs/1710.09207.

[26] Z.-H. Zhou and Ji Feng, "Deep forest: towards an alternative to deep neural networks," 2017, https://arxiv.org/abs/1702.08835.

[27] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, Chapman and Hall/CRC, New York, NY, USA, 2012.

[28] H. Jiang, Z. He, G. Ye, and H. Zhang, "Network intrusion detection based on PSO-XGBoost model," *IEEE Access*, vol. 8, no. 8, pp. 58392–58401, 2020.

[29] A. H. Lashkari, "Toward developing A systematic approach to generate benchmark android malware datasets and classification," in *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, IEEE, Montreal, QC, Canada, October 2018.