Malware Classification Based on Multilayer Perception and Word2Vec for IoT Security

YANCHEN QIAO, Cyberspace Security Research Center, Peng Cheng Laboratory
WEIZHE ZHANG, School of Computer Science and Technology, Harbin Institute of Technology
XIAOJIANG DU, Temple University
MOHSEN GUIZANI, Qatar University

With the construction of smart cities, the number of Internet of Things (IoT) devices is growing rapidly, leading to an explosive growth of malware designed for IoT devices. These malware pose a serious threat to the security of IoT devices. The traditional malware classification methods mainly rely on feature engineering. To improve accuracy, a large number of different types of features will be extracted from malware files in these methods. That brings a high complexity to the classification. To solve these issues, a malware classification method based on Word2Vec and **Multilayer Perception (MLP)** is proposed in this article. First, for one malware sample, Word2Vec is used to calculate a word vector for all bytes of the binary file and all instructions in the assembly file. Second, we combine these vectors into a 256x256x2-dimensional matrix. Finally, we designed a deep learning network structure based on MLP to train the model. Then the model is used to classify the testing samples. The experimental results prove that the method has a high accuracy of 99.54%.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; *Software reverse engineering*; *Malicious design modifications*; • **Computing methodologies** → Cross-validation;

Additional Key Words and Phrases: Malware classification, Word2Vec, multilayer perception, IoT

ACM Reference format:
Yanchen Qiao, Weizhe Zhang, Xiaojiang Du, and Mohsen Guizani. 2021. Malware Classification Based on Multilayer Perception and Word2Vec for IoT Security. *ACM Trans. Internet Technol.* 22, 1, Article 10 (September 2021), 22 pages.
https://doi.org/10.1145/3436751

Yanchen Qiao and Weizhe Zhang contributed equally to this research.

This work was supported in part by the Key-Area Research and Development Program of Guangdong Province (2019B010136001), the Basic and Applied Basic Research Major Program for Guangdong Province (2019B030302002), and the Science and Technology Planning Project of Guangdong Province (LZC0023 and LZC0024).

Authors' addresses: Y. Qiao, Cyberspace Security Research Center, Peng Cheng Laboratory, No. 2 Xingke 1st Street, Shenzhen, China, 518000; email: qiaoych@pcl.ac.cn; W. Zhang, School of Computer Science and Technology, Harbin Institute of Technology, No. 92, Xidazhi Street, Nangang District, Harbin, China, 150001, Cyberspace Security Research Center, Peng Cheng Laboratory, No. 2 Xingke 1st Street, Nanshan District, Shenzhen, China, 518000; email: wzzhang@hit.edu.cn; X. Du, Department of Computer and Information Sciences, Temple University, 1801 N. Broad Street, Philadelphia, USA, PA 19122; email: dxj@ieee.org; M. Guizani, Department of Compute Science and Engineering, Qatar University, University Street, Doha, Qatar; email: mguizani@ieee.org.

1533-5399/2021/09-ART10 $15.00
https://doi.org/10.1145/3436751

1 INTRODUCTION

Currently, as the Internet carries more information and property of users, the harm caused by cyber attacks is increasing. Especially with the construction of smart cities, the number of **Internet of Things (IoT)** [27, 29] devices is growing rapidly. The cyber attack aimed at IoT devices will actually affect people's lives and safety. The latest statistical result from AV-TEST [12] shows that more than 350,000 malware samples, which brings huge analysis and processing pressure to various security vendors and emergency departments, are increased every day. As the Advanced Persistent Threat attack gradually becomes the main trend, the time interval between attackers launching attacks and detecting attacks is getting longer. The traditional malware detection interval is usually in days, whereas the Advanced Persistent Threat attack is usually in months and years. An effective malware classification method can lessen the size of the virus database, improve the efficiency of malware detection, and discover new malware families. Therefore, there is an urgent need to develop an effective malware classification method to help security vendors identify and analyze new malware quickly and accurately.

Security vendors use anti-virus software to defend against malware to protect cyberspace. However, with the continuous application of new technologies, the security situation is constantly changing, revealing many new problems. Symantec once misreported the *netapi32.dll* and *lsasrv.dll* of some XP systems as *Backdoor.Haxdoor* and removed it, resulting in a large-scale system failure and a blue screen. Some anti-virus software misreports software that is packed or developed with easy language as a virus, which affects normal use. On May 12, 2017, *WannaCry* ransomware broke out globally, and at least 300,000 users were recruited in 150 countries, resulting in losses of $8 billion, affecting many industries such as finance, energy, and medical treatment. At the time of the incident, almost all of the anti-virus softwares were underreported. In 2015, after a comprehensive analysis of the malware of the *Duqu* family, Kaspersky was still successfully attacked into the company's intranet by Duqu 2.0. Afterward analysis found that the two have a large number of similar features and codes. In the early stage of the incident, Kaspersky failed to classify Duqu 2.0 effectively and was successfully attacked. These incidents indicate that the current mainstream malware detection and classification methods have false positives, false negatives, and problems of ineffective classification.

The new malware usually has two sources [4]. One is the completely innovative malware, which is developed by the creator and is different from older malware, and another, on the basis of existing malware, with the main purpose of avoiding detection, using new means of modifying part of the code, packing, confusion, and so forth to generate a new malware sample. The malware produced in the first way represents a new type of malware, which is harmful but has a low frequency. The malware produced in the second way is usually a variant of the existing malware family. The newly generated malware is mainly from the second way. Therefore, effectively and correctly classifying malware variants will help to aggregate new malware, reduce the amount of analysis work, and improve the defense effect.

The malware classification method mainly uses two types of features: dynamic and static. The dynamic characteristics are from the host behavior, network behavior, and other data that the malware runs in the sandbox, such as file operations, registry operations, mutex operations, process operations, domain name resolution, and URL requests. However, Chen et al. [40] found that more than 40% of the samples exhibited less malicious behavior under virtual machines. In addition, Katrenko [13] said that nearly 98% of modern malware used at least one evasive technique. Therefore, not all malware samples can run in virtual machines and have dynamic features extracted. Dynamic features often require the execution of malware files, resulting in a limited amount of analysis of malware per unit of time, greatly increasing the cost of extracting features. Static features mainly refer to binary, string, resource, timestamp, and so forth extracted on the basis of

malware binary file entities, and call flow diagrams, instruction sequences, API sequences, and so forth extracted based on the reversed assembly code. Static features are more general than dynamic features. However, the use of shelling, confusing, and other variant techniques leads to large differences in the same type of static eigenvalues extracted by different variants of the same family, resulting in false positives. Challenges are placed on features that rely on expert experience extraction.

The malware classification technology is always a research hotspot in the industry. The research points mainly focus on three aspects. First, *the feature selection*: the feature selection is mainly based on an expert experience to extract features combined with the technologies such as data mining, machine learning, and other methods. This method relies not only an expert experience but also the data, which makes the process complex. Second, the problem of *high feature dimension*: to avoid missing important features, the features are extracted as many as possible for classification model construction, resulting in high dimensionality and computational complexity. The third problem is *over-fitting*: due to the deviation of the data sample set, it usually leads to over-fitting and poor classification for some family samples.

To deal with these existing problems, we have done some research on malware and the neural network [18]. A malware classification method using Word2Vec and **Multilayer Perception (MLP)** is proposed in this article. Word2Vec [19], which was developed by Tomas Mikolov in 2013 at Google, is one of the most popular techniques for learning word embedding using the neural network. As we all know, there are a lot of bytes ranging from $0x00$ to $0xFF$ in hexadecimal form in a malware binary sample. When we treat bytes as words, each malware binary file could be regarded as a document written by bytes. An assembly file could be obtained by reversing the malware executable file using reverse tools. When we treat assembly instructions as words, each assembly file of a malware binary file could be considered as a document written by instructions. For a document, we could use Word2Vec to obtain word vectors of all words. Therefore, Word2Vec can also be used to train on the binary file. Then we use the trained model to obtain the word vectors of the bytes of malware binary samples and the assembly instructions of assembly files. Finally, for each sample, after connecting all the vectors together, we get a new feature vector. We use MLP to train the classification model on these new feature vectors. Then we classify the testing samples using the trained model. This method uses *no expert experience* and *no data dependence* in the feature extraction stage. Our method also avoids the over-fitting while reducing the feature dimension. The method is tested on the public sample set of BIG 2015 [31]. The results show that the method has a high accuracy.

The contributions of this article mainly focus on the following three aspects:

(1) We propose a malware feature representation method based on natural language processing methods, i.e., Word2Vec.
(2) We experimentally prove the word vector of binary bytes and assembly instructions in malware samples has significant differences between different malware families.
(3) We apply a deep learning algorithm to classify the vectorized malware, which achieves good results.
(4) We propose a malware classification method, which could detect IoT malware to guarantee the safety of IoT devices.

2 RELATED WORKS

Han et al. [10] proposed a method for converting the opcode sequence reversed from malware to RGB map using SimHash [3] and djb2 in 2013, in which SimHash is used to obtain the coordinate points in the graph. Finally, the method of random selection region matching is used to determine

whether they belong to the same family, and the accuracy of the method is 98.4%. In 2015, Shaid [34] proposed a method to visualize the malware behavior to a color map, expressing the risk level of different API calls in different colors, and each malware was formed into a color map. Ding and Zhu [6] proposed a malware detection method based on the Deep Belief Network in 2017. Based on the same n-gram assembly instruction sequence features, the accuracy of this method is significantly better than SVM, decision tree, and the K-nearest neighbor algorithm. In 2017, Kebede et al. [14] proposed a malware classification method using an automatic decoder based on a deep learning architecture. They converted the malware file into a 512x16 grayscale image, constructed a deep learning network using a multi-layer decoder and a Softmax layer, and input the training set to construct a classifier. The accuracy of the method was verified by experiments to be 99.15%.

Schultz et al. [33] proposed a malware variant detection method based on data mining in 2001. They used the static analysis method to extract the dynamic link library list from an import table, API list, API call frequency, strings, and byte sequences from the binary file, and then used the naive Bayes classifier for learning and classifying, and the accuracy rate reached 97.11%. Kolter and Maloof [16] improved the non-overlapping byte sequence to an n-gram byte sequence in 2004 and then used the decision tree to train the classifier. In 2008, Tian et al. [36] proposed a malware classification method using function length. They used the frequency of the function length (the number of bytes in the binary program) in the sample as a feature. Experiments show that the method could be fast and effective to classify Trojans. In 2012, Salehi et al. [32] used API in malware and API parameters as classification features, then used the dimensionality reduction method and multivariate classifier to classify malware. The test result showed that the accuracy rate reached 98.4%. Dahl et al. [5] proposed to use the stochastic prediction method to reduce the feature dimension, then used the neural network algorithm to achieve the classification of malware; the classification false positive was only 0.42%. Qiao et al. [24] proposed a malware binary file multi-channel visualization method, as well as a malware classification method based on deep learning. They use LeNet5 [11] to train the classification model. Qiao et al. [25] proposed a malware classification method based on the word vector of bytes in the malware sample and MLP.

Nari and Ghorbani [20] proposed a malware classification method based on network behavior in 2013. They obtained network packets by executing malware, built a network interaction flow chart based on data packets, and obtained the number of nodes, root out-degree, average out-degree, maximum out-degree from the graph, and other characteristics. Next, each sample could be converted into a vector, and finally they could use the classification algorithm to classify the malware. In 2013, Park et al. [21] proposed a method for detecting and classifying malware based on the common system call path. The experimental results showed that the false positive rate was close to 0%. In 2015, Pascanu et al. [22] regarded the instructions and APIs of malware dynamic execution as the language of malware, then used the cyclic neural network algorithm to classify malware, and the accuracy rate reached 98.3%. Giannella and Bloedorn [9] proposed a clustering method based on spectral clustering in 2015. The features used include dynamic information such as call graph of malware execution and HTTP. In 2016, Huang and Stokes [39] proposed a binary malware classification technology based on the multi-task deep learning framework, with 4.5 million samples as the training set and 2 million samples as the test set. The detection false-positive rate was only 0.358%, and the malware family classification error rate was only 2.94%.

Malware features mainly include dynamic and static. Dynamic features are mainly obtained by running samples in the sandbox. However, a lot of samples could not enter the main process [44], so the extracted dynamic features do not have unique family attributes. Therefore, although dynamic analysis can deal with code packing and confusion effectively (of which static methods have no such competence), its code coverage is not as good as static analysis. The static features used in the current classification methods mainly include import tables, strings, n-gram binary

sequences, APIs [41], assembly instructions, call graphs [38, 43], and so forth. Due to encryption and confusion, among others, features extraction is more difficult. The value of n in the n-gram is usually set to 2 or 3 in consideration of performance and does not well express the characteristics of the sample.

Word2Vec is a framework developed by Google's Mikolov et al. [19] to calculate word vectors. It is widely used in the study of text categorization [8], sentiment analysis [42], and other natural language processing [26, 28]. At present, there are also many works that Word2Vec and other natural language processing methods for malware detection and classification. Popov [23] proposed a malware detection method based on Word2Vec and machine learning in 2017. He regarded the assembly instruction sequence as a sentence in a document, treated a single assembly instruction as a word, and then used Word2Vec to calculate word vector of different instruction on the sample set. Next, he took the first n instructions of each sample, constructed each into a matrix, and finally constructed a classification model using a **Convolutional Neural Network (CNN)**. The test results showed that the method had 96% accuracy. In 2017, Tran and Sato [37] proposed a malware classification method based on Natural Language Processing and API. They first used dynamic analysis technology to obtain the API call sequence of malware, then used n-gram, Doc2Vec [17], TF-IDF, and other natural language processing methods to convert the API call sequence into a vector. Second, they constructed a classification model using SVM, K-nearest neighbor, MLP, the RF algorithm, and so forth. The test results showed that the accuracy was between 90% and 96%. Cakir and Dogdu [2] proposed a method of malware classification based on deep learning in 2018. The process of sample matrixing was similar to that of Popov [23] and then used GBM (Gradient Boosting Machine) [7] to construct a classification model; they achieved an accuracy of 96%.

An in-depth study of Word2Vec reveals that the word vector calculated by Word2Vec has a good ability to represent the language characteristics of the corpus. Under the same model, the word vectors calculated by the same word in the same category corpus are similar. However, in different categories, the word vectors calculated on the corpus differ greatly.

3 THEORETICAL BASIS

3.1 Word2Vec Principle

The early word vector is one-hot form, and it is the simplest one in the word vector forms—for a thesaurus, first counting V words contained in the dictionary, then fixing the order of the V words, and sorting according to letters, frequency, or order of appearance, and so forth. Finally, each word can be represented by a V-dimensional sparse vector, where only the position has the word 1, but all other positions are 0. Since the word vector is generally around 100,000, the dimension of the one-hot formal word vector will be large, which usually causes dimensionality disaster.

To solve the problem of one-hot word vector, a form of distributed word vector is proposed whose dimension is fixed and the values of all elements in the vector are arbitrary real numbers. At present, Word2Vec [19] is a kind of framework for calculating all the distributed word vectors. In this framework, the word vector is a by-product of the **Continuous Bag-of-Words (CBOW)** neural network model and Skip-gram. The model structure is shown in Figure 1. In the CBOW model, the input layer is the first n and the last n words of the word $w(t)$, the projection layer accumulates all the vectors of the input layer, and the output layer is a large binary tree containing all of the corpus. The initial value of all word vectors in the CBOW model can be a random specific dimension vector. The ultimate goal of training is to pass $w(t-n), w(t-n+1), \ldots, w(t-1), w(t+1), \ldots, w(t+n-1), w(t+n)$ to the model and output $w(t)$.

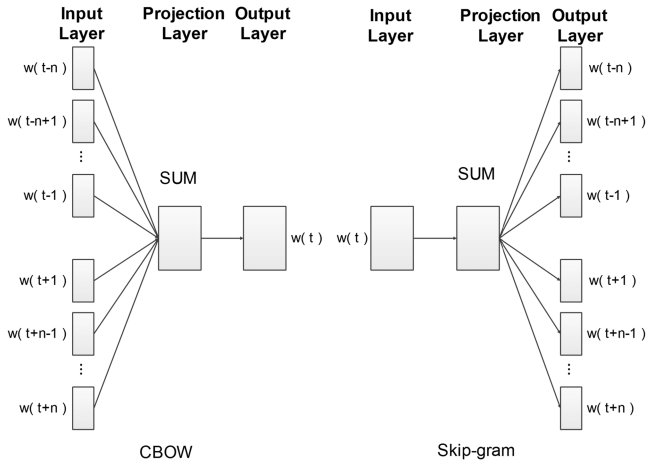

Fig. 1. CBOW and skip-gram of Word2Vec.

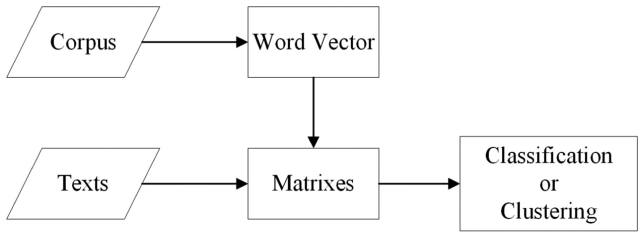

Fig. 2. General usage of Word2Vec.

3.2 Two Usages of Word2Vec

In general, the usage of Word2Vec is shown in Figure 2. The word vector of each word is calculated on a corpus, and the distance between vectors can be used to obtain the similar word set of each word. Then, the word vector is used for text matrixing. Finally, text clustering or labeling of the training classifier is based on the categories that the text already has. The advantage of this usage is that the word vector is calculated in advance. When converting text into a matrix, it is only necessary to combine the word vectors into a matrix in order. However, this method ignores the language characteristics of different categories of texts, and the language characteristics of different authors, functions, and types that have different language characteristics. All of these may lead to false positives in classifying and clustering.

In this article, the usage of Word2Vec is shown in Figure 3. Each text in the corpus is treated as a corpus. Word2Vec is used to calculate the word vector of each word appearing in the text. Then, a sequence of a fixed number of fixed orders is filtered out in the entire corpus. These sequences of words are then combined into a matrix according to the sequence of words and the word vectors calculated on the text. Finally, a classification or a clustering can be performed on the full corpus.

There are 10,868 files belonging to nine families in BIG 2015 [31]. First, Word2Vec was used to get the word vector of all bytes on the sample belong to BIG 2015. Then we calculated the distribution of the cosine similarity of the word vector of the bytes belonging to the same and different families.

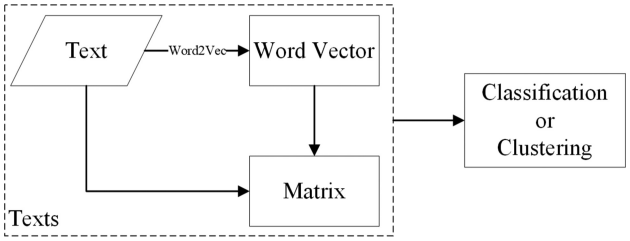

Fig. 3. The usage of Word2Vec in this article.

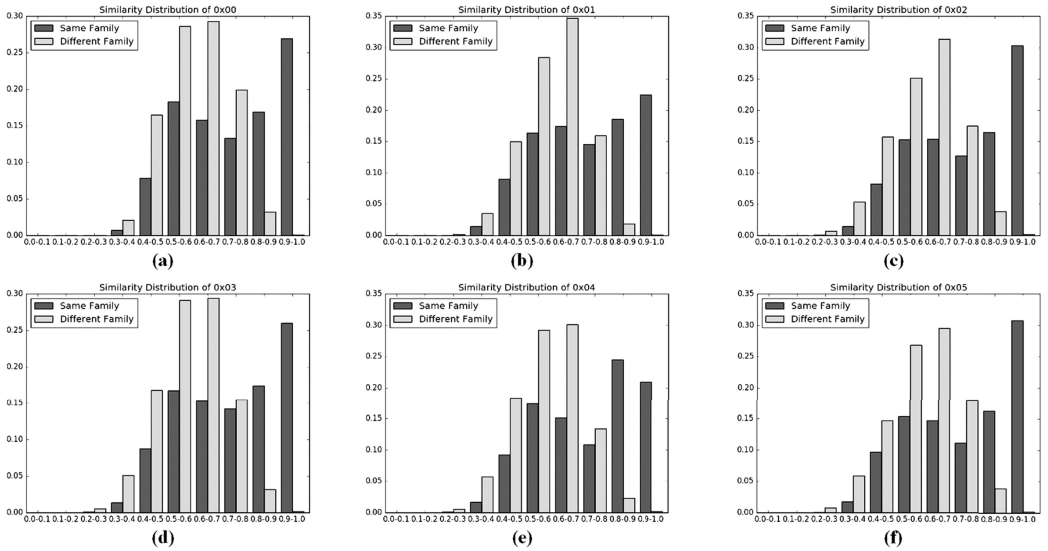

Fig. 4. The similarity distribution of word vectors of bytes of the same family and different family samples.

The result is shown in Figure 4. It can be seen from Figure 4 that the cosine similarity of the word vectors of $0x00$, $0x01$, $0x02$, $0x03$, $0x04$, $0x05$, and so on, of the same family sample is much larger than the cosine similarity of different family samples.

We also calculated the word vector of all assembly instructions on each sample, then the distribution of the cosine similarity of the word vector of the assembly instructions belonging to the same and different families. The result is shown in Figure 5. It can be seen from Figure 5 that the cosine similarity of the word vectors of *push*, *mov*, *pop*, *xor*, *call*, *cmp*, and so on, of the same family sample is much larger than the cosine similarity of different family samples.

The results in Figure 4 and Figure 5 show that word vector of bytes and assembly instructions calculated on each sample by Word2Vec has good family characteristics.

3.3 MLP

MLP is the most intuitive and simplest deep neural network. The basic structure is shown in Figure 6. An important feature is the full connection between the layers and the multi-layer structure. The first layer is the input layer, the middle layer is the hidden layer, and the last layer is the output layer. The hidden layer of MLP is not fixed, and the number of neurons in each layer of the hidden layer and the output layer is not fixed. In addition to the input layer, neurons of each

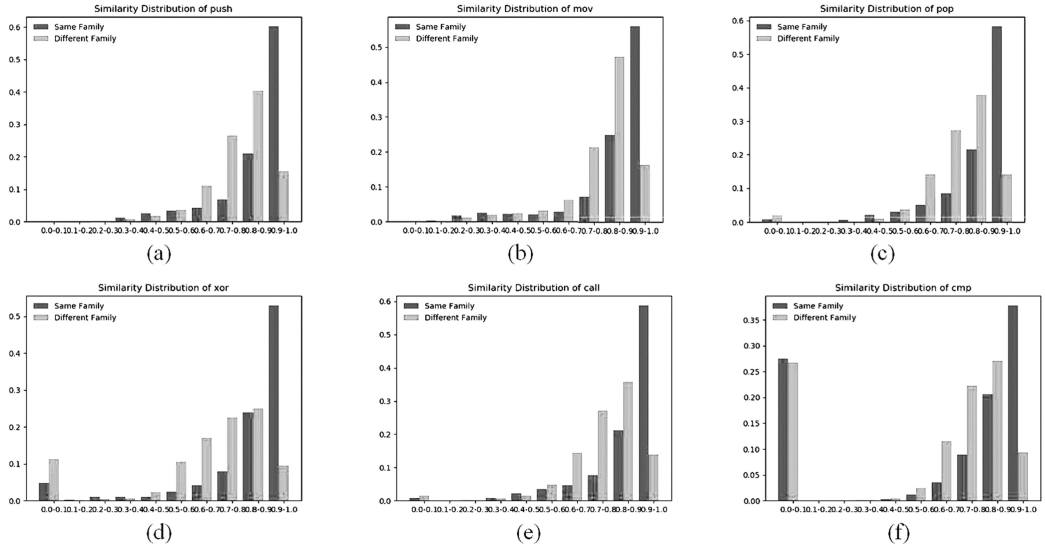

Fig. 5. The similarity distribution of word vectors of assembly instructions of the same family and different family samples.

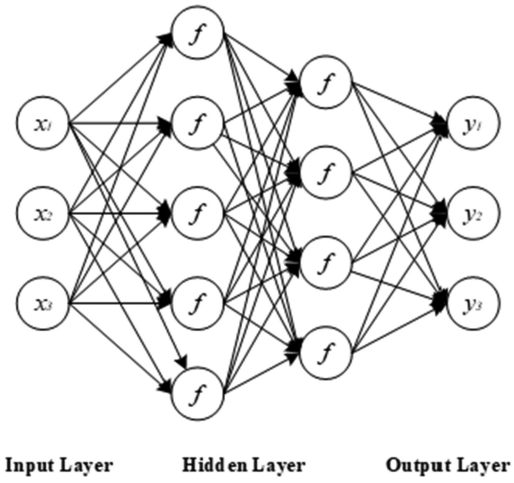

Fig. 6. The basic structure of MLP.

layer use an activation function. The ReLU function $relu(x) = max(0, x)$ generally is used as the activation function of the hidden layer, and the Softmax function $softmax(x_j) = \frac{x_j}{\sum_{k=1}^{K} x_k}$ is used as the activation function of the output layer.

Figure 7 shows the model of the neuron in the hidden layer of the Multilayer Perceptron. The output of the neuron is $y = f(\sum_{i=1}^{K} w_i \cdot x_i + b) = f(WX + b)$, where w_i represents weights, W represents weight matrix, x_i represents input, X represents the input matrix, b represents bias, and $f(\cdot)$ represents activation function.

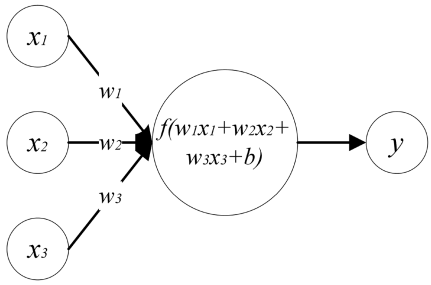

Fig. 7. The model of the neuron in the hidden layer of MLP.

With forward propagation, the output of the five neurons in the first layer of the hidden layer in Figure 6 is as follows:

$$\begin{cases} y_1^1 = f(u_1^1) = f(\sum_{i=1}^{K} w_1^{1i} \cdot x_i + b_1^1) = f(w_1^{11}x_1 + w_1^{12}x_2 + w_1^{13}x_3 + b_1^1) \\ y_1^2 = f(u_1^2) = f(\sum_{i=1}^{K} w_1^{2i} \cdot x_i + b_1^2) = f(w_1^{21}x_1 + w_1^{22}x_2 + w_1^{23}x_3 + b_1^2) \\ y_1^3 = f(u_1^3) = f(\sum_{i=1}^{K} w_1^{3i} \cdot x_i + b_1^3) = f(w_1^{31}x_1 + w_1^{32}x_2 + w_1^{33}x_3 + b_1^3) \\ y_1^4 = f(u_1^4) = f(\sum_{i=1}^{K} w_1^{4i} \cdot x_i + b_1^4) = f(w_1^{41}x_1 + w_1^{42}x_2 + w_1^{43}x_3 + b_1^4) \\ y_1^5 = f(u_1^5) = f(\sum_{i=1}^{K} w_1^{5i} \cdot x_i + b_1^5) = f(w_1^{51}x_1 + w_1^{52}x_2 + w_1^{53}x_3 + b_1^5). \end{cases} \tag{1}$$

The subscript indicates the layer where it is located, and the superscript indicates the location of the neuron. For example, y_1^1 indicates the output of the first neuron in the first layer. Then the preceding expression is converted into a matrix:

$$Y_1 = \begin{bmatrix} y_1^1 \\ y_1^2 \\ y_1^3 \\ y_1^4 \\ y_1^5 \end{bmatrix} = f\left(\begin{bmatrix} w_1^{11} & w_1^{12} & w_1^{13} \\ w_1^{21} & w_1^{22} & w_1^{23} \\ w_1^{31} & w_1^{32} & w_1^{33} \\ w_1^{41} & w_1^{42} & w_1^{43} \\ w_1^{51} & w_1^{52} & w_1^{53} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_1^2 \\ b_1^3 \\ b_1^4 \\ b_1^5 \end{bmatrix} \right) = f(W_1 \cdot X + B_1). \tag{2}$$

Y_1 represents the output of the first layer, W_1 represents the weight matrix of the first layer, and B_1 represents the offset matrix of the first layer. To extend the forward propagation calculation process of layer 1 to any layer in the network, such as layer l, then

$$\begin{cases} y_l^j = f(u_l^j) \\ u_l^j = \sum_{i \in L_{l-1}} w_l^{ji} y_{l-1}^i + b_l^j \\ Y_l = f(U_l) = f(W_l y_{l-1} + B_l). \end{cases} \tag{3}$$

L_l represents the neuron of the layer l, the output of this layer is Y_l, the output of the j-th neuron of this layer is y_l^j, and the input of this node is u_l^j. The weight matrix connecting layer 1 and layer $l-1$ is W_l, and the weight of the i-th node of layer $l-1$ to the j-th node of layer l is w_l^{ji}.

After the network structure of the MLP is completed, all that is done during training is to use the back propagation of the loss from the back network layer to the front network layer to complete the parameter update. In this work, the MLP is used for classification. There are multiple neurons in the output layer, and each neuron corresponds to a classification. Let the input sample be $X = [x_1, x_2, \ldots, x_n]$ and its classification is $Y = [y_1, y_2, \ldots, y_m]$. According to Equation (3), for the last layer of the network (assumed to be the k-th layer), namely the output layer, the loss function is

defined as

$$L(Y, Y_k) = \sum_{i=1}^{m} l(y_i, y_{ki}). \tag{4}$$

To minimize the loss function, it is derived by gradient

$$\begin{cases} \frac{\partial L}{\partial w_l^{ji}} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial w_l^{ji}} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial u_l^j} \cdot \frac{\partial u_l^j}{\partial w_l^{ji}} \\ \frac{\partial L}{\partial b_l^j} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial b_l^j} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial u_l^j} \cdot \frac{\partial u_l^j}{\partial b_l^j}. \end{cases} \tag{5}$$

According to the previous definition, $\frac{\partial y_l^j}{\partial u_l^j} = f'(u_l^j)$, $\frac{\partial u_l^j}{\partial w_l^{ji}} = y_{l-1}^i$, $\frac{\partial u_l^j}{\partial b_l^j} = 1$, and therefore Equation (5) can be converted to

$$\begin{cases} \frac{\partial L}{\partial w_l^{ji}} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial w_l^{ji}} = \frac{\partial L}{\partial y_l^j} \cdot f'(u_l^j) \cdot y_{l-1}^i \\ \frac{\partial L}{\partial b_l^j} = \frac{\partial L}{\partial y_l^j} \cdot \frac{\partial y_l^j}{\partial b_l^j} = \frac{\partial L}{\partial y_l^j} \cdot f'(u_l^j). \end{cases} \tag{6}$$

According to the network structure, the input of the neuron in layer $l+1$ is the output of layer l, and hence the loss function can be regarded as the function of the input of each neuron in layer $l+1$, then

$$\frac{\partial L}{\partial y_l^j} = \sum_{k \in L_{l+1}} \frac{\partial L}{\partial y_{l+1}^k} \cdot \frac{\partial y_{l+1}^k}{\partial u_{l+1}^k} \cdot w_{l+1}^{kj}. \tag{7}$$

The sensitivity of the node is defined as the rate of change of the error to the input: $\delta = \frac{\partial L}{\partial u}$, then the sensitivity of the j-th neuron in the layer l is

$$\delta_l^j = \frac{\partial L}{\partial u_l^j} = \frac{\partial L}{\partial y_l^j} f'(u_l^j) = f'(u_l^j) \sum_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{kj}. \tag{8}$$

Therefore, the gradient of the loss function to each parameter is

$$\begin{cases} \frac{\partial L}{\partial w_l^{ji}} = \frac{\partial L}{\partial u_l^j} \cdot \frac{\partial u_l^j}{\partial w_l^{ji}} = \delta_l^j y_{l-1}^i \\ \frac{\partial L}{\partial b_l^j} = \frac{\partial L}{\partial u_l^j} \cdot \frac{\partial u_l^j}{\partial b_l^j} = \delta_l^j. \end{cases} \tag{9}$$

To express all nodes of each layer in a matrix way, the update formula of parameters of each layer is

$$\begin{cases} W_l := W_l - \eta \frac{\partial L}{\partial W_l} = W_l - \eta \delta_l Y_{l-1}^T \\ B_l := B_l - \eta \frac{\partial L}{\partial B_l} = B_l - \eta \delta_l. \end{cases} \tag{10}$$

$\eta \in R$ is the learning rate. The network parameter update depends on the learning rate. The larger the learning rate, the larger the parameter update step, and the smaller the learning rate, the smaller the parameter update step.

4 CLASSIFICATION METHOD

A malware classification method based on Word2Vec and MLP is proposed in this article. First, for one malware sample, Word2Vec is used to calculate a word vector for all bytes of the binary file and all instructions in the assembly file. Second, we combine these vectors into a 256x256x2-dimensional matrix. Finally, we designed a deep learning network structure based on MLP to train the model.

```
FF 90 F2 6E FF 8F F2 6D FF 92 F5 72 FF 93 F4 72
FF 92 F5 72 FF 92 F5 71 FF 92 90 97 FF 82 72 88
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 01 00 01 00 10 10 00 00 01 00 20 00 68
04 00 00 16 00 00 00 28 00 00 00 10 00 00 00 20
00 00 00 01 00 20 00 00 00 00 00 00 04 00 00 12
0B 00 00 12 0B 00 00 00 00 00 00 00 00 00 00 24
27 AC FF 24 32 A6 FF 26 48 A6 FF 2C 58 A7 FF 34
5A A3 FF 3B 54 9F FF 3F 49 9E FF 3E 3A 9E FF 34
34 9E FF 1C 3E A0 FF 00 4A AB FF 00 3F D2 FF 00
```

Fig. 8. Bytes in one malware binary file.

4.1 Word Vector of Bytes

There is an enormous number of bytes ranging from $0x00$ to $0xFF$ in hexadecimal form in a malware binary sample, as shown in Figure 8.

We see many consecutive $0xCC$ and consecutive $0x00$ in the malware binary sample. In executable files, especially PE files, the byte $0xCC$ usually indicates an interrupt instruction, whereas consecutive $0xCC$ is usually used for alignment. Continuous $0x00$ is mainly used to separate sections in the executable file. In this work, we treat 256 bytes as words used for writing the document, and the same consecutive bytes are treated as delimiters, thus cutting a document into multiple sentences. In the long-term malware analysis work, five or more consecutive $0x00$ or $0xCC$ are meaningless bytes. After removing them from the binary file, each malware example can be converted into a document comprised of many sentences written by bytes.

Then the Word2Vec algorithm is used to train the document converted from a malware binary file. For each byte, we could get a word vector. In this work, the dimension of the word vector is set to be 256, and hence the word vector of bytes can be represented as

$$W_S^i = (x_1^i, x_2^i, \ldots, x_{256}^i), i \in [0x00, 0xff]. \tag{11}$$

In the preceding equation, W_S^i represents the i-th byte's word vector in the malware file S. Then a 256x256 matrix M_S^{Byte} could be converted by combining the word vectors of all 256 bytes in ascending order, as follows. This method was first proposed by Qiao et al. [25].

$$M_S^{Byte} = \begin{bmatrix} W_S^{0x00} \\ W_S^{0x01} \\ W_S^{0x02} \\ \vdots \\ W_S^{0xff} \end{bmatrix} = \begin{bmatrix} x_1^{0x00} & \cdots & x_{256}^{0x00} \\ \vdots & \ddots & \vdots \\ x_1^{0xff} & \cdots & x_{256}^{0xff} \end{bmatrix} \tag{12}$$

The values in the word vector calculated by the Word2Vec algorithm are mostly decimal and negative, as well as the values in M_S^{Byte}. We want to confirm again at the image level whether this method has obvious class discrimination. Hence, we need to normalize M_S^{Byte}, as follows:

$$M'^{Byte}_S = int\left(\frac{M_S^{Byte} - \min(M_S^{Byte})}{\max(M_S^{Byte}) - \min(M_S^{Byte})} \times 255\right). \tag{13}$$

We convert the matrix M'^{Byte}_S to a grayscale image, as shown in Figure 9. As we all can see from the figure, there is a significant difference among the grayscale images converted from

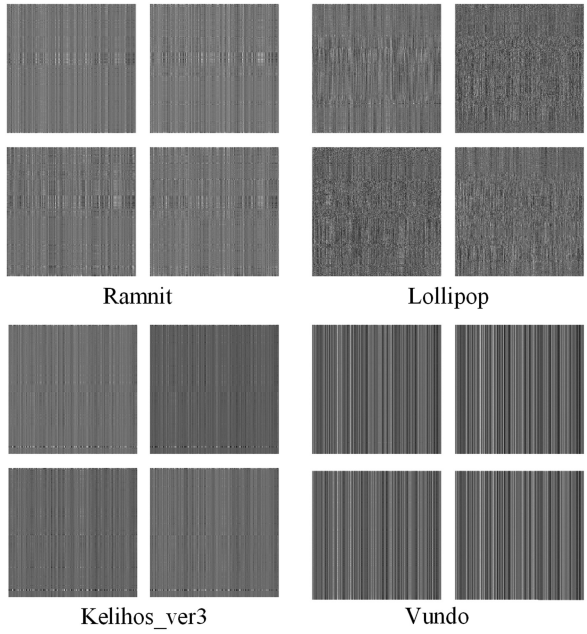

Fig. 9. Grayscale images converted from word vectors of bytes.

different malware families, whereas the similarity among the grayscale images converted from the same malware family is very high. This proves that the word vector of byte has obvious class discrimination.

4.2 Word Vector of Assembly Instructions

The assembly file could be reversed from the executable malware sample. An assembly file consists of many functions, which starts with "proc near/far" and ends with "endp". A function is typically comprised of several code blocks separated by "loc_xxx", as shown in Figure 10.

We have found in a lot of malware analysis work that for a function, even if the same compilation tool is used and the same compilation options are set, if the function's position in the source code is different, the function's jump position in the assembly code will also different. To eliminate the effects of position differences, the constant parameters in the assembly file are deleted, and only the assembly instruction sequence is retained, as in the method of Shankarapani et al. [35].

In summary, the assembly code documentation process is as follows:

- We reverse a executable file to an assembly file.
- The assembly file is split into functions according to "proc near/far" and "endp".
- The assembly instruction sequence are extracted from the function by removing parameters.
- An article written by assembly instructions is converted from the assembly file of a malware sample.

We collected 213 commonly used assembly instructions, such as mov, push, xor, and cmp. They are sorted alphabetically and represented as $I = \{ins_1, ins_2, \ldots, ins_{213}\}$. Then the word vectors of assembly instructions are calculated using the CBOW model of Word2Vec, for all malware samples. The dimension of the word vector is also set to be 256. The word vector of an assembly instruction

```
sub_402010      proc near

Src           = dword ptr  4

      mov     edx, [esp+Src]
      push    esi
      mov     esi, ecx
      mov     eax, edx
      push    edi
      mov     dword ptr [esi+18h], 0Fh
      mov     dword ptr [esi+14h], 0
      mov     byte ptr    [esi+4], 0
      lea     edi, [eax+1]
      nop

loc_402030:
      mov     cl, [eax]
      inc     eax
      test    cl, cl
      jnz     short loc_402030
      sub     eax, edi
      push    eax
      push    edx
      mov     ecx, esi
      call    sub_43E75E
      pop     edi
      mov     eax, esi
      pop     esi
      retn    4
sub_402010      endp
```

Fig. 10. Example of a function in an assembly file.

is represented as $W_s^{ins_i} = [x_1^{ins_i}, x_2^{ins_i}, \ldots, x_{256}^{ins_i}], ins_i \in I$, where $W_S^{ins_i}$ indicates the word vector of the assembly instruction ins_i in the malware file S.

All word vectors of the instructions of one malware file are combined into a matrix in alphabetical order. There may be some instructions that some samples do not contain. For them, a 256-dimensional zero vector, denoted as $\vec{0}$, is filled. We fill 43 (256 − 213 = 43) zero vectors in the end. Finally, every assembly file of malware is converted into a matrix, as follows. This method was first proposed by Qiao et al. [24].

$$M_S^{Ins} = \begin{bmatrix} W_S^{ins_1} \\ \vdots \\ W_S^{ins_{213}} \\ \vec{0}^{214} \\ \vdots \\ \vec{0}^{256} \end{bmatrix} = \begin{bmatrix} x_1^{ins_1} & \cdots & x_{256}^{ins_1} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \tag{14}$$

We also want to confirm again at the image level whether this method has obvious class discrimination. Hence, we need to normalize M_S^{Ins} as follows:

$$M'^{Ins}_S = int\left(\frac{M_S^{Ins} - \min(M_S^{Ins})}{\max(M_S^{Ins}) - \min(M_S^{Ins})} \times 255\right). \tag{15}$$

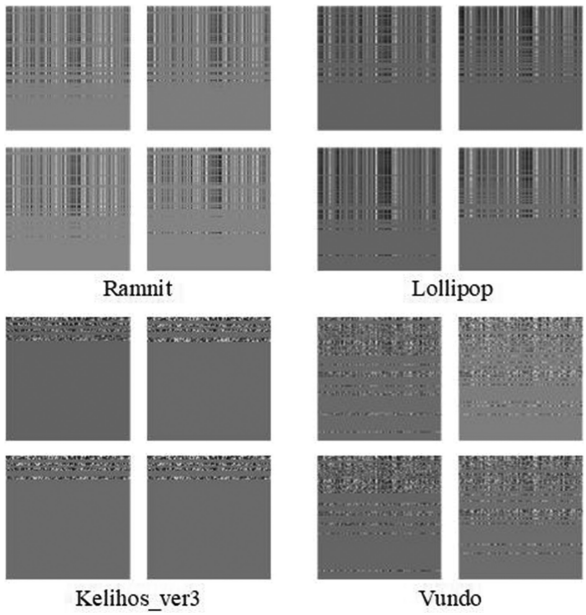

Fig. 11. Grayscale images converted from word vectors of assembly instructions.

We convert the matrix M'^{Ins}_{S} into a grayscale image, as shown in Figure 11. As we all can see from the figure, there is a significant difference among the grayscale images converted from different malware families, whereas the similarity among the grayscale images converted from the same malware family is very high. This proves that word vector of assembly instruction has obvious class discrimination.

4.3 Deep Learning Network Structure

After the preceding two steps, for each malware sample, the matrixes M^{Byte}_{S} and M^{Ins}_{S} are combined into a 256x256x2-dimensional matrix M_S.

$$M_S = [M^{Byte}_{S}, M^{Ins}_{S}] \tag{16}$$

The structure of the neural network in this article is designed based on MLP adding the DROPOUT. The details of the network structure are shown in Figure 12, and each layer's functions are as follows: *INPUT*: Input layer; each malware is pre-processed, converted into a matrix of 256x256x2 dimensions, and expanded into a vector of 131,072 dimensions, so the input dimension is 131,072.

FC1: Fully connected layer; this layer includes 512 units, and using the ReLU activation function, each unit is fully connected with the input feature vector. DROPOUT is used to reduce over-fitting, and the probability of DROPOUT sets to 0.2.

FC2: Fully connected layer; this layer includes 512 units, and using the ReLU activation function, each unit is fully connected with the feature vector output by the FC1 layer. DROPOUT is used to reduce over-fitting, and the probability of DROPOUT sets to 0.2.

OUTPUT: Output layer; this layer is fully connected with the previous layer FC2. The length of the output corresponds to the total number of classes. We use Softmax as the classification function.

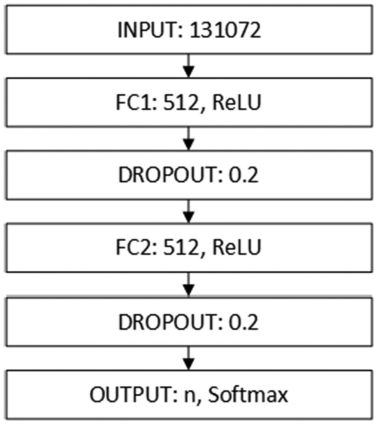

Fig. 12. Deep network structure of this work.

Table 1. Experimental Results of This Method

Order	Precision	Recall	*F*1 Score
1	99.27%	99.26%	99.25%
2	99.26%	99.26%	99.26%
3	99.54%	99.54%	99.53%
4	99.08%	99.07%	99.07%
5	98.88%	98.80%	98.80%
6	99.01%	98.98%	98.98%
7	97.96%	97.78%	97.77%
8	99.18%	99.17%	99.16%
9	99.09%	99.07%	99.07%
10	98.82%	98.70%	98.73%

5 EXPERIMENTAL EVALUATION

5.1 Experiment on Traditional Malware

5.1.1 Traditional Malware Dataset. In this work, the dataset BIG 2015 [31] is used for experiments. The Microsoft Malware Classification Challenge was announced in 2015 along with a publication of a huge dataset, consisting of disassembly and bytecode of more than 20K malware samples. The dataset has become a standard benchmark for research on modeling malware behavior. The training set of BIG 2015 contains 10,868 samples belonging to nine families. There are two types of files for each sample in the dataset: a binary file and an assembly file. This work uses both files.

5.1.2 Experimental Results. This work uses 10-fold cross validation for experiments. We scramble the entire dataset and then divide it into 10 subsets to ensure that each subset contains samples of each family. In each experiment, one of them is used as the test set, one is used as the verification set, and the other eight are combined as the training set. Table 1 shows the results of 10 experiments. As we all can see from Table 1, the average precision rate of 10 experiments is 99.01%. This shows that the precision of our method is very high.

We put the detailed results from one experiment in Table 2. In this experiment, the accuracy rate is 99.54%. Then we draw the confusion matrix for this experiment.

Table 2. The Results of One Experiment

Malware Family	Precision	Recall	F_1 Score	Support
Ramnit	0.9938	0.9938	0.9938	161
Lollipop	1.0000	1.0000	1.0000	264
Kelihos_ver3	1.0000	1.0000	1.0000	260
Vundo	1.0000	0.9756	0.9877	41
Simda	1.0000	0.7500	0.8571	4
Tracur	0.9875	0.9875	0.9875	80
Kelihos_ver1	1.0000	1.0000	1.0000	35
Obfuscator.ACY	0.9760	0.9919	0.9839	123
Gatak	1.0000	1.0000	1.0000	113
Avg./total	0.9954	0.9954	0.9953	1,081

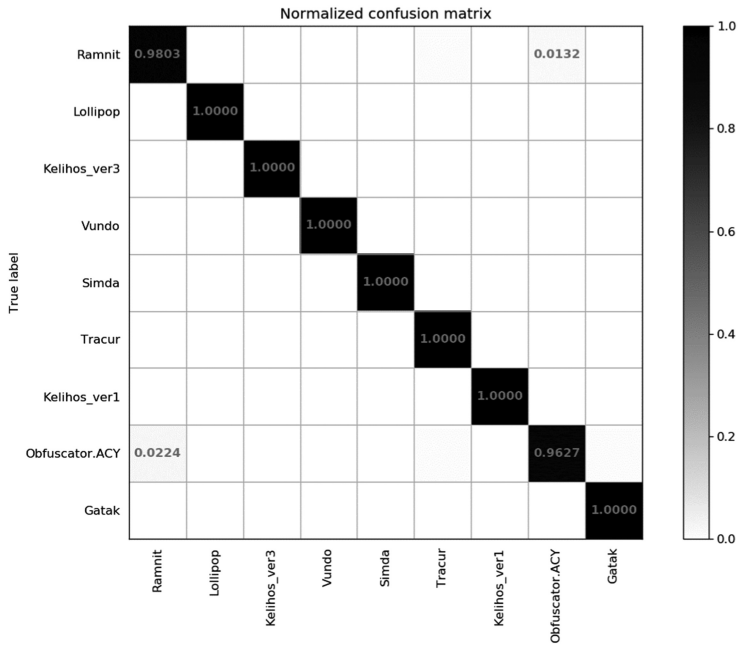

Fig. 13. Confusion matrix for multi-classification using MLP model.

The confusion matrix of the experiment for the MLP multi-class classifier is shown in Figure 13. The blocks represent the scores of malware samples belonging to the malware family on the vertical axis, and they are classified into malware families on the horizontal axis, where 0 represents white and 1 represents black. As can be seen from Figure 13, almost all malware families have achieved high classification accuracy.

Figure 14 shows the ROC curve for multi-classification of the method in this article. It can be seen that the area under the curve is up to almost 1.0000.

5.2 Experiment on IoT Malware

5.2.1 IoT Malware Dataset. Thousands of IoT malware files, which were in **Executable and Linkable Format (ELF)**, were collected and were labeled by VirusTotal [1]. VirusTotal inspects

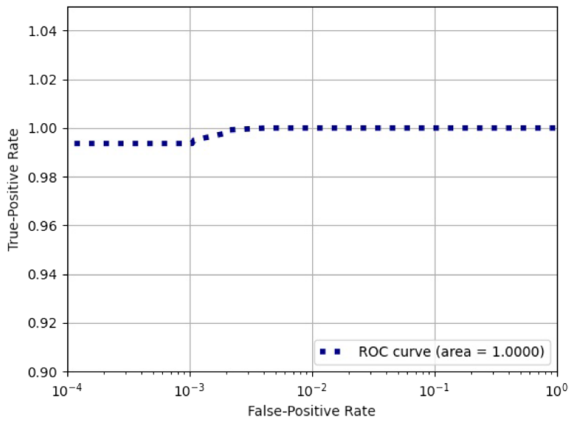

Fig. 14. ROC curve for multi-classification using the MLP model.

Table 3. Details of the IoT Malware Dataset

Family Name	Family Tag	Number of Samples
Mirai	0	4,688
Gafgyt	1	2,817
Tsunami	2	383
Ganiw	3	228
Dofloo	4	205
Mayday	5	137

items with more than 70 anti-virus scanners and URL/domain blacklisting services, in addition to myriad tools to extract signals from the studied content. The results are widely accepted by scientific research and industry. Among them, we selected 8,458 files from six IoT malware families, as shown in Table 3. For the binary ELF file, we used reverse tools to get its assembly code. Finally, we have the binary ELF file and assembly file for every malware sample.

5.2.2 Experimental Results. We also use 10-fold cross validation for experiments on IoT malware. As we all can see from Table 4, the average precision rate of 10 experiments is 98.41%. The precision is very high, and it is basically equal to the results of experiments on traditional malware, shown in Table 1.

We put the results of all classes of one experiment in Table 2. It can be seen from Table 5 that the accuracy rate of five IoT malware families is 96.67%, the average recall rate is 96.57%, and the average $F1$ score is 96.49%.

The confusion matrix for the MLP multi-class classifier on IoT malware is shown in Figure 15. As can be seen ine Figure 15, almost all malware families achieve high classification accuracy.

Figure 16 shows the ROC curve for multi-classification on IoT malware of the method in this article. It can be seen that the area under the curve is 0.9979.

5.3 Work Comparison

Kim [15] proposed a method for analyzing images through artificial intelligence deep learning in 2018, and protecting big data by quickly detecting malware. Kim [15] also performed experiments on the training set of the Microsoft Malware Classification Challenge (BIG 2015) [31], and the highest accuracy was 91.76%. Rahul et al. [30] also converted malware binary files into grayscale

Table 4. Experimental Results of This Method on IoT Malware

Order	Precision	Recall	$F1$ Score
1	96.67%	96.57%	96.49%
2	99.39%	97.63%	97.57%
3	99.94%	97.98%	97.92%
4	94.17%	97.51%	97.49%
5	99.64%	96.80%	96.69%
6	99.28%	96.80%	96.72%
7	99.56%	98.34%	98.28%
8	99.63%	96.56%	96.46%
9	97.86%	97.51%	97.57%
10	98.19%	97.04%	96.97%

Table 5. Results of One Experiment on IoT Malware

Malware Family	**Precision**	**Recall**	***F*1 Score**	**Support**
Mirai	0.9716	0.9917	0.9816	483
Gafgyt	0.9593	0.9557	0.9575	271
Tsunami	1.0000	0.7000	0.8235	30
Ganiw	1.0000	0.8462	0.9167	26
Dofloo	0.8571	1.0000	0.9231	24
Mayday	1.0000	1.0000	1.0000	12
Weighted avg.	0.9667	0.9657	0.9649	846

images and then used deep learning methods for classification. They also tested on the BIG 2015 [31] dataset, and the average accuracy rate reached 94.91%. The results of the work comparison are shown in Table 6. Among the three works, the work of this article has the highest accuracy.

5.4 Discussion

Based on the preceding two experiments, our method's average classification accuracy for traditional malware reached 99.54%, and the average classification accuracy for IoT malware was as high as 98.41%. They show that this method not only effectively classifies traditional malware but also accurately classifies IoT malware. This work is an extension of our previous work [25]. The previous work only used the features based on the word vector of bytes in the malware binary file, and its performance on IoT malware is slightly worse. This work is also an extension of our other previous work [24]. In this work, another byte feature is used, and experiments have shown that this feature has no positive effect on the classification results. In addition, this previous work uses the CNN, and it has been shown that the CNN does not improve accuracy but reduces efficiency.

The results of the work's comparison with our previous works are shown in Table 7. Among the three works, the work of this article has the highest accuracy and better efficiency.

6 CONCLUSION AND FUTURE WORK

As the types of features used in current malware classification methods continue to increase, the difficulty of feature extraction is increasing. As a result, the complexity of classification becomes

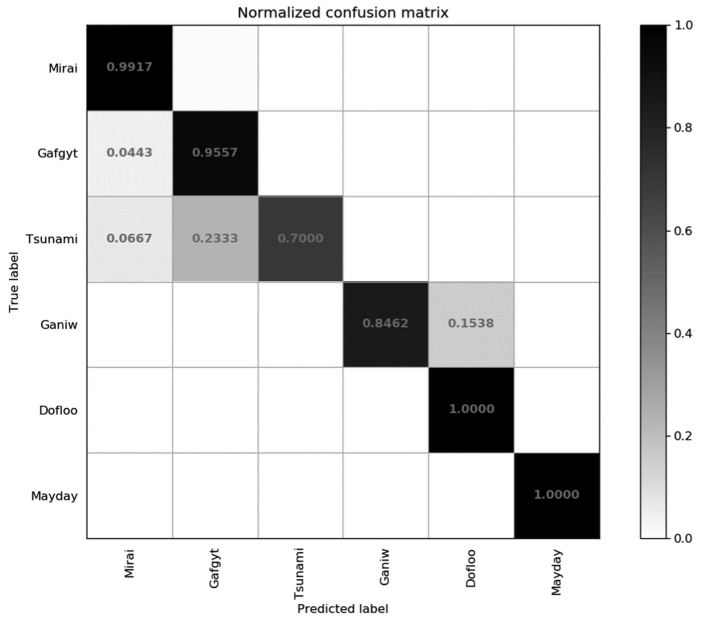

Fig. 15. Confusion matrix for multi-classification on IoT malware using the MLP model.

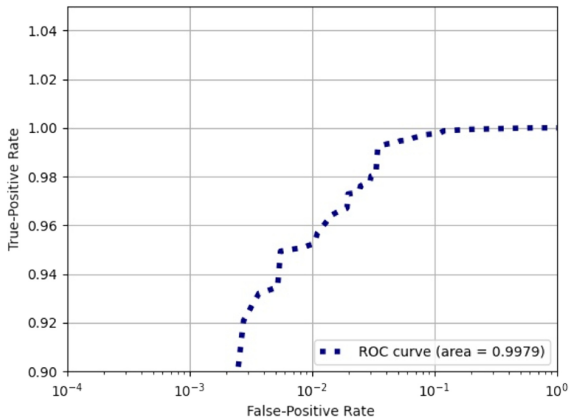

Fig. 16. ROC curve for multi-classification on IoT malware using the MLP model.

Table 6. Work Comparison

Word	Feature	Method	Accuracy
Kim [15]	File grayscale	CNN	91.76%
Rahul et al. [30]	File grayscale	CNN	94.91%
This work	Word vectors of bytes and assembly instructions	MLP	99.54%

higher and higher. However, the accuracy has not improved significantly. Malware designed for IoT devices poses a serious threat to the security of IoT devices. A malware classification method based on word vector of bytes and MLP is proposed in this article. The key idea of this method is that the relationship of bytes and assembly instructions in the same family sample is similar,

Table 7. Comparison with Our Previous Works

Work	Feature	Method	Accuracy
Previous work 1 [24]	Multi-channel image	LeNet5	98.76%
Previous work 2 [25]	Word vectors of bytes	MLP	98.89%
This work	Word vectors of bytes and assembly instructions	MLP	99.54%

and the relationship of bytes and assembly instructions in different family samples is distinctly different. Therefore, the word vector of bytes and assembly instructions has the ability to describe the characteristics of the malware family. It is feasible to classify malware families based on word vectors of bytes and assembly instructions. IoT malware could also be accurately detected and classified. This will greatly improve the efficiency of malware analysts and guarantee the safety of IoT devices. With the construction of smart cities, the number of IoT devices is growing rapidly, leading to an explosive growth of malware designed for IoT devices. These malware pose a serious threat to the security of IoT devices. In the future, we will conduct in-depth research on IoT malware detection and classification.

REFERENCES

[1] Bernardo Quintero, Emiliano Martínez, Víctor Manuel Álvarez, Karl Hiramoto, Julio Canto, Alejandro Bermúdez, and Juan A. Infantes. 2020. VirusTotal. Retrieved July 29, 2021 from https://www.virustotal.com/.

[2] Bugra Cakir and Erdogan Dogdu. 2018. Malware classification using deep learning methods. In *Proceedings of the ACMSE 2018 Conference (ACMSE'18)*. Article 10, 5 pages. https://doi.org/10.1145/3190645.3190692

[3] Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th ACM Symposium on Theory of Computing*.

[4] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. 2020. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 987–1001.

[5] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, Los Alamitos, CA, 3422–3426.

[6] Yuxin Ding and Siyi Zhu. 2017. Malware detection based on deep learning algorithm. *Neural Computing & Applications* 1 (2017), 1–12.

[7] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[8] Jin Gao, Yahao He, Xiaoyan Zhang, and Yamei Xia. 2017. Duplicate short text detection based on Word2vec. In *Proceedings of the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS'17)*.

[9] Chris Giannella and Eric Bloedorn. 2015. Spectral malware behavior clustering. In *Proceedings of the 2015 IEEE International Conference on Intelligence and Security Informatics (ISI'15)*. IEEE, Los Alamitos, CA, 7–12.

[10] Kyoung Soo Han, Jae Hyun Lim, Eul Gyu Im, Kyoung Soo Han, Jae Hyun Lim, and Eul Gyu Im. 2013. Malware analysis method using visualization of binary files. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS'13)*. 317–321.

[11] Simon Haykin and Bart Kosko. 2009. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.

[12] AV-TEST Institute. 2020. Malware Statistics & Trends Report. Retrieved July 29, 2021 from http://www.av-test.org/en/statistics/malware/.

[13] Anna Katrenko. 2020. Malware Sandbox Evasion: Techniques, Principles & Solutions. Retrieved July 29, 2021 from https://www.apriorit.com/dev-blog/545-sandbox-evading-malware.

[14] T. M. Kebede, O. Djaneye-Boundjou, B. N. Narayanan, A. Ralescu, and D. Kapp. 2017. Classification of malware programs using autoencoders based deep learning architecture and its application to the Microsoft malware classification challenge (BIG 2015) dataset. In *Proceedings of the 2017 IEEE National Aerospace and Electronics Conference (NAECON'17)*. 70–75. https://doi.org/10.1109/NAECON.2017.8268747

[15] Hae Jung Kim. 2018. Image-based malware classification using convolutional neural network. In *Advances in Computer Science and Ubiquitous Computing*. Lecture Notes in Computer Science, Vol. 474. Springer, 1352–1357. https://doi.org/10.1007/978-981-10-7605-3_215

[16] Jeremy Z. Kolter and Marcus A. Maloof. 2004. Learning to detect malicious executables in the wild. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, 470–478.

[17] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning*. 1188–1196.

[18] G. Lin, S. Wen, Q. L. Han, J. Zhang, and Y. Xiang. 2020. Software vulnerability detection using deep neural networks: A survey. *Proceedings of the IEEE* 108, 10 (2020), 1825–1848.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781.

[20] Saeed Nari and Ali A. Ghorbani. 2013. Automated malware classification based on network behavior. In *Proceedings of the 2013 International Conference on Computing, Networking, and Communications (ICNC'13)*. IEEE, Los Alamitos, CA, 642–647.

[21] Younghee Park, Douglas S. Reeves, and Mark Stamp. 2013. Deriving common malware behavior through graph clustering. *Computers & Security* 39 (2013), 419–430.

[22] Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *Proceedings of the 2015 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'15)*. IEEE, Los Alamitos, CA, 1916–1920.

[23] Igor Popov. 2017. Malware detection using machine learning based on Word2Vec embeddings of machine code instructions. In *Proceedings of the 2017 Siberian Symposium on Data Science and Engineering (SSDSE'17)*. IEEE, Los Alamitos, CA, 1–4.

[24] Yanchen Qiao, Qingshan Jiang, Zhenchao Jiang, and Liang Gu. 2019. A multi-channel visualization method for malware classification based on deep learning. In *Proceedings of the 2019 18th IEEE International Conference on Trust, Security, and Privacy in Computing and Communications and the 13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'19)*. IEEE, Los Alamitos, CA, 757–762.

[25] Y. Qiao, B. Zhang, and W. Zhang. 2020. Malware classification method based on word vector of bytes and multilayer perception. In *Proceedings of the 2020 IEEE International Conference on Communications (ICC'20)*. IEEE, Los Alamitos, CA, 1–6.

[26] Youyang Qu, Longxiang Gao, Tom H. Luan, Yong Xiang, Shui Yu, Bai Li, and Gavin Zheng. 2020. Decentralized privacy using blockchain-enabled federated learning in fog computing. *IEEE Internet of Things Journal* 7, 6 (2020), 5171–5183.

[27] Youyang Qu, Shui Yu, Longxiang Gao, Wanlei Zhou, and Sancheng Peng. 2018. A hybrid privacy protection scheme in cyber-physical social networks. *IEEE Transactions on Computational Social Systems* 5, 3 (2018), 773–784.

[28] Youyang Qu, Shui Yu, Jingwen Zhang, Huynh Thi Thanh Binh, Longxiang Gao, and Wanlei Zhou. 2019. GAN-DP: Generative adversarial net driven differentially privacy-preserving big data publishing. In *Proceedings of the IEEE International Conference on Communications (ICC'19)*. IEEE, Los Alamitos, CA, 1–6.

[29] Youyang Qu, Shui Yu, Wanlei Zhou, Sancheng Peng, Guojun Wang, and Ke Xiao. 2018. Privacy of things: Emerging challenges and opportunities in wireless Internet of Things. *IEEE Wireless Communications* 25, 6 (2018), 91–97.

[30] R. K. Rahul, T. Anjali, Vijay Krishna Menon, and K. P. Soman. 2017. Deep learning for network flow analysis and malware classification. In *Proceedings of the International Symposium on Security in Computing and Communication*. 226–235.

[31] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft malware classification challenge. arXiv:1802.10135.

[32] Zahra Salehi, Mahboobeh Ghiasi, and Ashkan Sami. 2012. A miner for malware detection based on API function calls and their arguments. In *Proceedings of the 2012 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP'12)*. IEEE, Los Alamitos, CA, 563–568.

[33] Matthew G. Schultz, Eleazar Eskin, F. Zadok, and Salvatore J. Stolfo. 2001. Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P'01)*. IEEE, Los Alamitos, CA, 38–49.

[34] Syed Zainudeen Mohd Shaid. 2015. Malware behavior image for malware variant identification. In *Proceedings of the International Symposium on Biometrics and Security Technologies*. 238–243.

[35] Madhu K. Shankarapani, Subbu Ramamoorthy, Ram S. Movva, and Srinivas Mukkamala. 2011. Malware detection using assembly and API call sequences. *Journal in Computer Virology* 7, 2 (2011), 107–119.

[36] Ronghua Tian, Lynn Margaret Batten, and S. C. Versteeg. 2008. Function length as a tool for malware classification. In *Proceedings of the 2008 3rd International Conference on Malicious and Unwanted Software (MALWARE'08)*. IEEE, Los Alamitos, CA, 69–76.

[37] Trung Kien Tran and Hiroshi Sato. 2017. NLP-based approaches for malware classification from API sequences. In *Proceedings of the 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES'17)*. IEEE, Los Alamitos, CA, 101–105.

[38] Huanran Wang, Hui He, and Weizhe Zhang. 2018. Demadroid: Object reference graph-based malware detection in Android. *Security and Communication Networks* 2018 (2018), Article 7064131.
[39] Wenyi Huang and Jack W. Stokes. 2016. MtNet: A multi-task neural network for dynamic malware classification. In Detection of Intrusions and Malware, and Vulnerability Assessment. Lecture Notes in Computer Science, Vol. 9721. Springer, 399–418. https://doi.org/10.1007/978-3-319-40667-1_20
[40] Xu Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. 2008. In *Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN'08)*. IEEE, Los Alamitos, CA.
[41] Bin Zhang, Wentao Xiao, Xi Xiao, Arun Kumar Sangaiah, Weizhe Zhang, and Jiajia Zhang. 2020. Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes. *Future Generation Computer Systems* 110 (2020), 708–720.
[42] Dongwen Zhang, Hua Xu, Zengcai Su, and Yunfeng Xu. 2015. Chinese comments sentiment classification based on word2vec and SVMperf. *Expert Systems with Applications* 42, 4 (2015), 1857–1863.
[43] W. Zhang, H. Wang, H. He, and P. Liu. 2020. DAMBA: Detecting Android malware by ORGB analysis. *IEEE Transactions on Reliability* 69, 1 (2020), 55–69.
[44] W. Zhang, B. Zhang, Y. Zhou, H. He, and Z. Ding. 2020. An IoT honeynet based on multi-port honeypots for capturing IoT attacks. *IEEE Internet of Things Journal* 7, 5 (2020), 3991–3999.

Received May 2020; revised October 2020; accepted November 2020