Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Recurrent neural network for detecting malware

Sudan Jha[a], *Deepak Prashar*[a], *Hoang Viet Long*[b,c,*], *David Taniar*[d]

[a] *Computer Science and Engineering, Lovely Professional University, Punjab, India*
[b] *Division of Computational Mathematics and Engineering, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam*
[c] *Faculty of Mathematics and Statistics, Ton Duc Thang University, Ho Chi Minh City, Vietnam*
[d] *Faculty of Information Technology, Monash University, Melbourne, Australia*

ARTICLE INFO

Article history:
Received 26 March 2020
Revised 26 July 2020
Accepted 6 September 2020
Available online 10 September 2020

Keywords:
Area under the curve (AUC)
Recurrent neural network (RNN)
Malware detection
Text classification
Word2Vec

ABSTRACT

In this paper, we propose an efficient Recurrent Neural Network (RNN) to detect malware. RNN is a classification of artificial neural networks connected between nodes to form a directed graph alongside with a temporal sequence. In this paper, we have conducted several experiments using different values of hyper parameters. From our rigorous experimentations, we found that the step size is a more important factor than the input size when using RNN for malware classification. To justify the proof-of-concept for RNN as an efficient approach for malware detection, we measured the performance of RNN with three different feature vectors using hyper parameters. The three feature vectors are "hot encoding feature vector", "random feature vector" and "Word2Vec feature vector". We also performed a pairwise *t*-test to test the results if they are significant with each other. Our results show that, RNN with Word2Vec feature vector achieved the highest Area Under the Curve (AUC) value and a good variance among three feature vectors. From the empirical analysis, we conclude that RNN with feature vectors pertained by the Skip-gram architecture of Word2Vec model is best for malware detection with high performance and stability.

1. Introduction

Malware is a program developed with malicious intent and has become a big cyber threat around the world. There are a lot of methods to detect malware. Generally, signature-based methods are widely used for detecting malware. It detects the malware by using a signature that is collected from detected malwares in the past. But the disadvantage of this method is that it is very hard to detect unseen or modified malware. Due to this, researchers are widely using behavior-based and machine learning methods. The behavior-based method detects malware by observing the program in a specific environment, like sandbox. At the same time, we see that the digital technique is getting improved and malware is also evolving in various ways to avoid being detected. Therefore, more complex and fine methods, such as machine learning, are required. Machine Learning techniques, especially Deep Learning is an excellent technique that deals with variants of data because it can not only learn the given feature but also automatically extract features from data to achieve the goal of classification task. In case of malware detection, this method will help in classifying whether a particular file is malware or not. The

* Corresponding author at: Division of Computational Mathematics and Engineering, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam.
E-mail address: hoangvietlong@tdtu.edu.vn (H.V. Long).
https://doi.org/10.1016/j.cose.2020.102037

most recent researches have shown that deep learning techniques are applied to detect malware.

In this paper, we use an RNN model to effectively detect malware. As the first step, we carefully investigate feature vectors for texts (documents) and used three techniques of NLP (Natural Language Processing): one hot encoding, random vector representation, and trained vector representation using a Word2Vec model, followed by several experiments in order to find the influence of hyper parameters on RNN network (e.g. step size, input size, etc). As the second step, we perform experiments with RNN with the above-mentioned three feature vectors. We will discuss the performance and stability of them, and propose an efficient method with RNN for malware detection.

'Random feature vectors' and 'Word2Vec feature vectors' use different random seeds; whereas, one hot encoding feature vectors use different vocabulary dictionary. This paper signifies the performance between feature vectors by conducting rigorous experiments for each feature vector.

We performed a paired *t*-test with 95% confidence level and found that the Area Under the Curve (AUC) value of each pair is significant. The result with F1 score was same as the AUC value. But in our study and from the experiments we performed, the random method produced the largest variance in terms of accuracy.

Machine learning-based solutions have been already used as a tool to supersede signature-based anti-malware systems. Andrade et al. (2019) described that malicious and legitimate samples used in RNN to estimate statistical difference inorder to create adversarial examples, and they are better than that of the signature-based methods. Similarly, the vulnerability of machine learning algorithms in malware detection is evaluated by classifying this algorithm using a discriminant function on a set of data points. However, from our results, we see that this eventually gives a higher misclassification rate.

In Word2Vec, the 'continuous skip-gram' weighs nearby context words more heavily than more distant context words. While the order still is not captured, each of the context vectors is weighed and compared independently. But in our proposed work, it weighs against the average context. The results show that RNN with the Word2Vec feature vector has the highest Area Under the Curve (AUC) value and gives good variance among any other feature vectors. This results in a very high performance and produces stability. Thus, it makes our proposed model best fit for malware detection.

Novelty: From the analytical results of all the vector approaches, we found that the Word2Vec feature vector has the highest value of average and significant variance. Therefore, we recommend the trained Word2Vec feature vector with small input size to malware detection using RNN, and this is the novelty of our work.

Contributions: Our major contribution in this work is the extensive use of Word2vec model. We applied the Word2Vec model in featured vectors, natural language processing, encoding, random and trained vector representation to check and validate the influence of hyper parameters like step size and input size on RNN networks. Our rigorous experiments using the Word2Vec model indicate a better performance and stability. None of previous researchers have been successful in using Word2Vec to achieve this degree of performance and stability. Therefore, we conclude that our method is the most efficient method for detecting malware using RNN.

2. Related work

In this section, we first provide a brief discussion of malware detection techniques, with an emphasis on feature extraction, families of malware, Word2Vec, classifiers; which are the basis for the proposed work presented in this paper. We also review relevant related work for malware detection using recurrent neural network. Finally, we discuss AUC, which give us a convenient means to quantify the various experiments that we have conducted. There are many approaches to the malware detection problem. In this section, we briefly consider signature-based, behavior-based, and statistical-based detection.

Static data, derived directly from code, can be collected quickly. Though signature-based methods fail to detect obfuscated or entirely new malware, researchers have extracted other features for static detection. Saxe and Berlin (2015) distinguish malware from benign ware using a deep feed-forward neural network with a true-positive rate of 95.2% using features derived from code. However, the true-positive rate falls to 67.7% when the model is trained using files only seen before a given date and tested using those discovered for the first time after that date, indicating the weakness of static methods in detecting completely new malwares.

Hiai and Shimada (2019) have correlated RNN with relation vector. In their work, they have developed relation information between pairs of role expressions, such as "boss and staff," and propose a sarcasm detection method based on surface and relation information.

Long et al. (2020) presented an efficient algorithm and tool to detect web security vulnerabilities (which can also be treated as malware). Their experimental results are capable of detecting vulnerabilities with high accuracy than any other previously exisiting methods. Wetzker et al. (2010) presented an approach to trend detection in social bookmarking systems using a probabilistic generative model in combination with smoothing techniques. Social bookmarking systems are gaining major interest among researchers in the areas of data mining and Web intelligence, since they provide a large amount of user-generated annotations and reflect the interest of millions of people.

Damodaran et al. (2017) conducted a comparative study of static, behavioural and hybrid detection models for malware detection and found behavioural data to give the highest area under the curve (AUC) value, 0.98, using Hidden Markov Models with a dataset of 785 samples. Additionally, Grosse et al. (2016) show that, in the case of Android software, static data can be obfuscated to cause a classifier previously achieving 97% accuracy to fall as low as 20% when classifying obfuscated samples. Training using obfuscated samples allowed a partial recovery of accuracy, but accuracy did not improve beyond random chance.

Methods using dynamic data assume that malware must enact the behaviors necessary to achieve their aims. Typically, these approaches capture behaviours such as API calls to the operating system kernel.

Table 1 – Number of malwares used in experiment according to Malware Family.

Malware Family	Number of malwares
Ramnit	729
Lollipop	1178
Kelihos_ver3	1365
Vundo	224
Simda	21
Tracur	349
Kelihos_ver1	149
Obfuscator.ACY	546
Gatak	439
Total	5000

Tobiyama et al. (2016) use RNNs to extract features from 5 min of API call log sequences which are then fed into a convolutional neural network to obtain 0.96 AUC score with a dataset of 170 samples.

Firdausi et al. (2010) compare machine learning algorithms trained on API calls and achieve an accuracy of 96.8% using correlation-based feature selection and a J48 decision tree. The 250 benign samples used for the experiment are all collected from the WindowsXP System32 directory, which is likely to give a higher degree of homogeneity than benign software encountered in the wild.

Ahmed et al. (2009) detect malware using API call streams and associated metadata with a Naive Bayes classifier, achieving 0.988 AUC, again with the 100 benign samples being WindowsXP 32-bit system files. Tian et al. (2010) and use Random Forests trained on API calls and associated metadata to achieve 97% accuracy and a 98% F-Score respectively.

Huang and Stokes (2016) achieve the highest accuracy in the literature, 99.64%, using System API calls and features derived from those API calls using a shallow feed-forward neural network. Huang and Stokes (2016) and Pascanu et al. (2015) are outliers with much larger datasets, both obtained through access to the corpus of samples held privately by the authors' companies. The majority of research does not mention a time-cap on file execution, in these cases we may presume that the files are executed until activity stops. The median data capture time frame for those reported is 5 min (see Table 1).

2.1. *Feature extraction*

Feature Extraction is a way to transform data into representative features. It is important to determine the representation of input data as it can affect model's performance and computational cost. In a malware detection domain, there are several ways to do feature extraction. Ranveer and Hiray (2015) categorized features based on malware analysis by static analysis and dynamic analysis. Schultz et al. (2001) used "system resource information", "strings" and "byte sequences" with three different algorithms to compare the results obtained from the proposed data mining methods and the traditional signature-based method. Veeramani and Rai (2012) used Windows APIs (Application Programming Interfaces) with various values of *n*-gram and achieved high detection performance with an SVM classifier. Siddiqui et al. (2008) used frequency of instruction sequences occurrence. Ahn et al. (2017) proposed a method that used both opcode and Windows API calls and showed a better performance than using each feature itself. In this paper, we used only the opcode as a feature and extraction method. The flowchart of feature extraction of our experiment is shown in Fig. 1.

In Fig. 1 the grey box is an example of data transformation. In case of benign data, we needed to disassemble using the pefile and capstone modules in Python. After disassembling, the data is converted to string. Then, we extracted the opcode string by matching with the already defined opcode list. The defined opcode list contains the x86-Assembler mnemonics. If no opcode is found in the defined opcode list, then it will be ignored. After feature extraction, each file is represented with opcode sequences.

In general, malware can be represented by two common ways, which are the "hex view" and the "assembly view" Ranveer and Hiray (2015). In the hex view, machine codes are represented as a sequence of hexadecimal digits. The hexadecimal digits contain the sequence of acquisition of 16-bytes words. In 2015, Microsoft held a malware classification challenge Kaggle (2015). Microsoft Malware Classification Challenge, which was announced in 2015, provided a huge dataset of nearly 0.5 terabytes, consisting of disassembly and bytecode of more than 20K malware samples. This dataset contains 8 different families of malware, which are "Ramnit", "Lollipop", "Kelihos_ver3", "Vundo", "Simda", "Tracur, Kelihos_ver1", "Obfuscator ACY" and "Gatak". Each malware file has a 20 characters' hash ID that uniquely identifies the file and class. Microsoft also provided each file with two types of extension, which are '.byte' and '.asm'. The ".byte" has the file's binary content and is represented in hexadecimal. But this extension doesn't have a PE header (Portable Executable header) to ensure the sterility. On the other hand, The ".asm" is generated by the Interactive Disassembler tool (IDA) and contains various metadata information extracted from the binary.

2.2. *Word2Vec*

Machine Learning techniques have been used quite extensively in the field of NLP (Natural Language Processing). There are several techniques such as one hot encoding, BOW (Bag of Words), TF-IDF (Text Frequency-Inverse Document Frequency), and vector representation. Examples of these methods are shown in Fig. 2.

The one hot encoding method represents the word with a vector which length is number of words in the vocabulary. Each word has a unique vector that is the value of 1 (i.e. the index of word in the vocabulary), or 0 otherwise. For example, in Fig. 2, 'push' has a '0' index in the vocabulary. Therefore, it can be represented by '1 0 0'. Thus, a document can be represented by matrix, which is number of words in the document by number of words in the vocabulary.

The BOW method represents the document whether it has the word or not as Boolean value or frequency of each word occurring in the document. For example, in Fig. 2 which used frequency value, the document contains 1 'push', 2 'mov' and 1 'xor'. Through the vocabulary index, the document can be represented by '1 2 1'. Then, a document can be represented by vector which length is number of words in vocabulary.

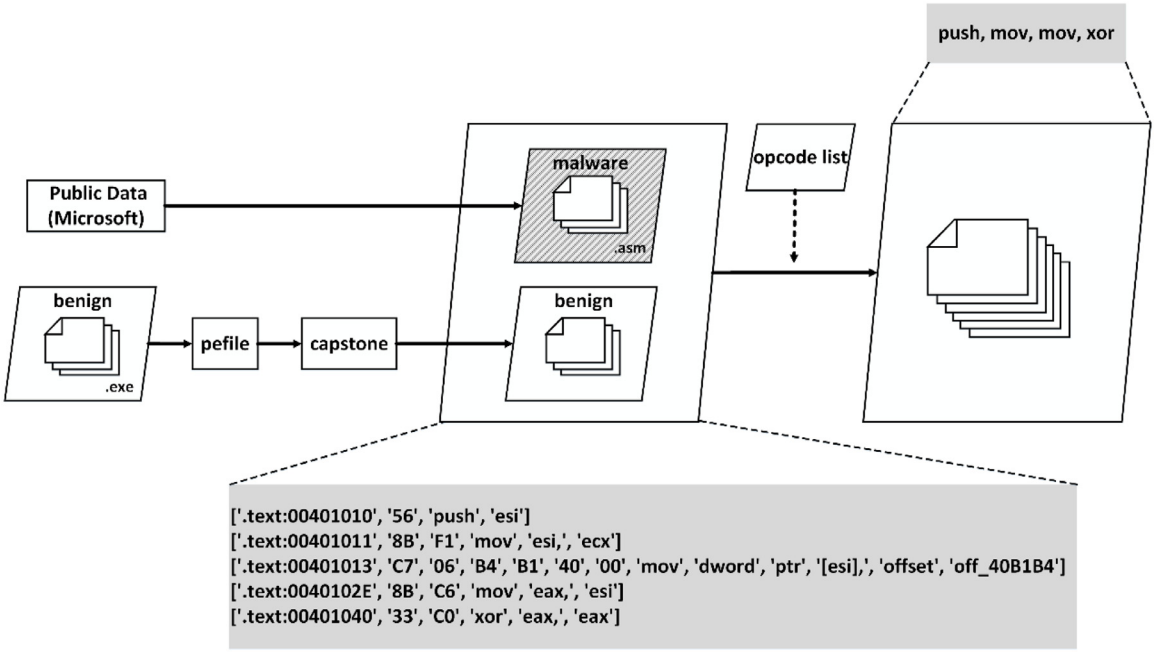

Fig. 1 – The flowchart of feature extraction.

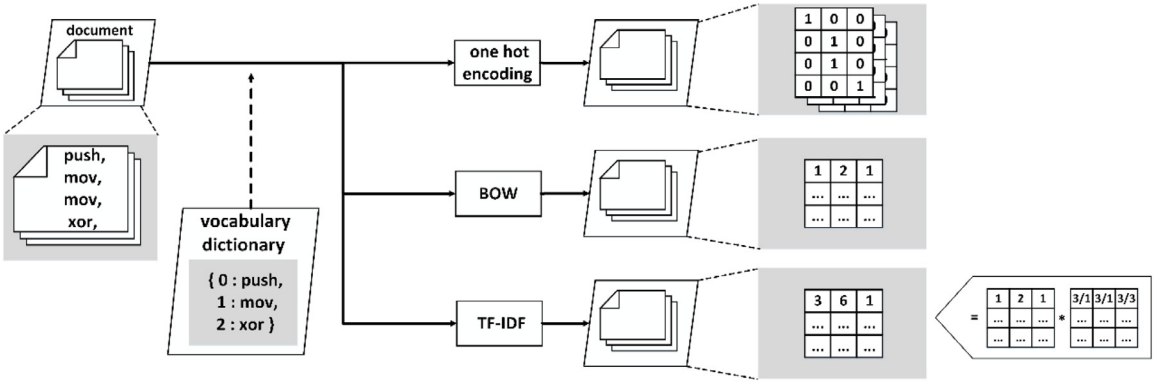

Fig. 2 – Example according to NLP methods.

The TF-IDF method is a weighted version of BOW. This method is one of the very important methods because the importance of each word by frequency of word in whole documents is evaluated. TF-IDF multiplies TF (Text Frequency, i.e. the frequency of the term in the document) and IDF (Inverse Document Frequency, i.e. total number of documents divided by number of document that the term is occurred). There are several formulas, but the main idea is that rare words are more important than common words. For example, in Fig. 2, TF is same with result of the BOW method. In case of 'push', we set the IDF value to 3/1 meaning that the total number of documents is 3 and the number of documents that contain 'push' is 1. After doing same operation in every word in the document, we can see that the TF-IDF value of 'push' is higher than 'xor', even they have same TF value. We know the reason that 'push' is rarer than 'xor'. However, those three methods do not consider the order and the relationship of words.

The Word2Vec model generates vector representation of each word in the document. Based on the distributional hypotheses, the Word2Vec model assumes that the word which is occurred in a similar context has similar meaning. Consequently, word embedding techniques are used widely in various domains. Popov (2017) used this technique with CNN classifier for malware detection. The embedding vectors are initialized randomly and trained to predict the context of words. Therefore, vector space can learn to capture relationship between words, like syntactic and semantic information.

Word2Vec can be divided in two types, according to the input and target of neural network (Mikolov et al., 2013). The first is the CBOW (Continuous Bag of Words) model and the second is the Skip-gram model. In the CBOW model, its input is the average of context words and target is the center word. In the Skip-gram, its input is the center word and target is each context word. The number of context words is user-defined.

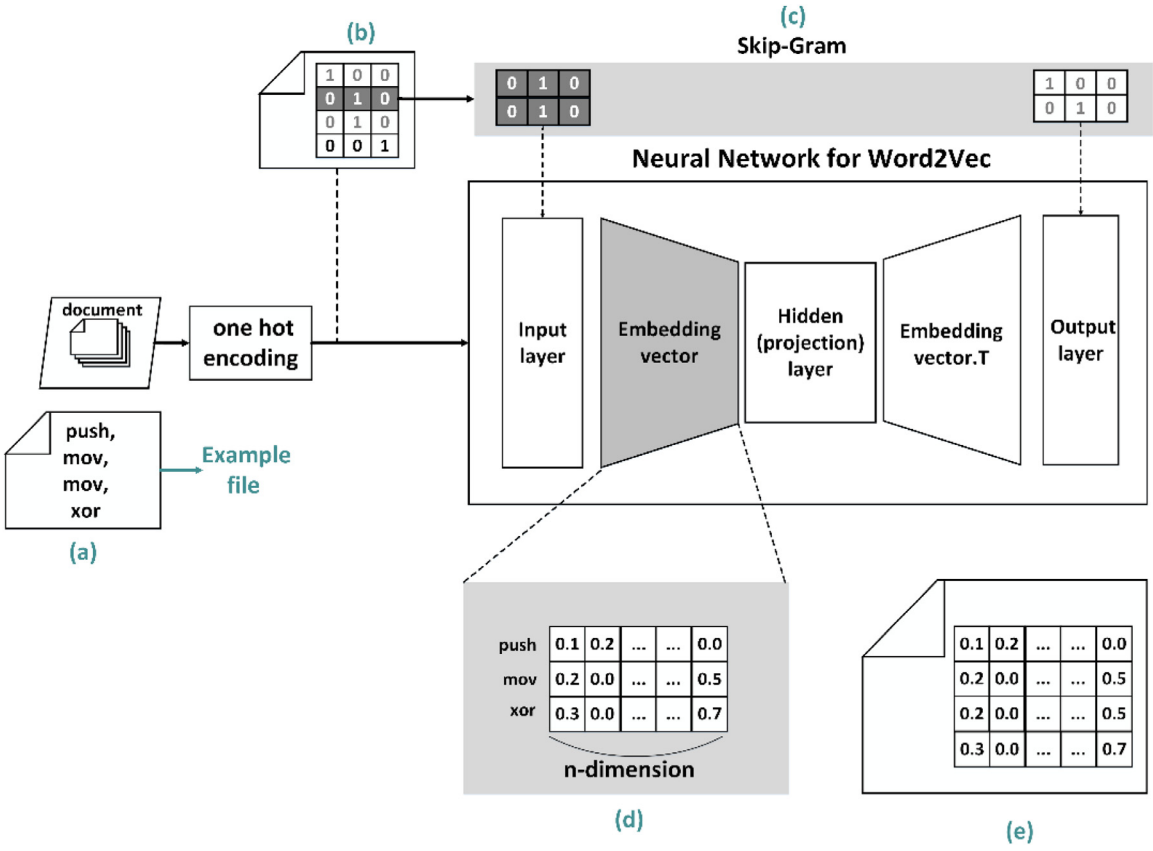

Fig. 3 – Working progress of the Word2Vec model (Skip-gram).

Both neural networks contain input layer, one hidden layer called the projection and output layers. The weights between input layer and hidden layer is the embedding vector, and the dimension is determined by number of neurons in the hidden layer. The weights between hidden and output are transformed from the former weights. The working progress of neural network of the Word2Vec model is depicted in Fig. 3.

2.3. *Classifier*

Within the Deep Learning domain, there are a few representative models, such as CNN (Convolution Neural Network) and RNN (Recurrent Neural Network). The CNN model is widely used in image classification domain, since it captures the low-level features in low layers and assembles them into high-level features by the convolution operation. Gibert (2016) used CNN for malware classification by converting binary data into grey-scale images. As we already mentioned, (Popov, 2017) used a CNN model which is referred by Kim (2014) with word embedding technique for malware detection.

RNN is widely used with sequence data, since it has feedback connections between past and present in recurrent neurons, which is also called a cell. This connection makes the model learn sequential pattern of data according to time step. However, with basic recurrent cells, when the number of time step is large, it exhibits a gradient problem that the memories of first inputs can gradually fade away. Therefore, an LSTM (Long Short-Term Memory) cell is introduced for preventing memory loss. The architecture of the LSTM is shown in Fig. 4.

Unlike recurrent basic cell, LSTM cell has two different state vectors, which are $h(t)$ for the short-term state, and $c(t)$ for the long-term state (Géron, 2017). Also, there are 4 distinguishing operations: forget gate, input gate, memory cell, and output gate. The main layer of cell is memory cell. It operates with the present input and previous short-term state, just like basic recurrent neuron. Also, it has extra operations with other gates' output and can affect both short-term state and long-term state. The forget gate determines which memory is ignored or remained from the long-term state and current input. The input gate determines which memory is added to the long-term state. The output gate determines which parts of the long-term state should be read and output at the present tie step. Because of these gates, LSTM cell can keep previous information and use it to make better decisions.

The classification flowchart of the RNN model with an LSTM cell is shown in Fig. 5.

In Fig. 5, the grey box is an example of data transformation. Firstly, the opcode which is the text data is converted to a numerical value by an index of the vocabulary dictionary. For example, 'push' is converted into 0, because it is the first word of in the vocabulary. Then the index values are mapped with each embedding vector and represented to vector value. In the example, 'push' is converted to [0.1, 0.2] because it is the first value of the embedding vector. Note that the vocabulary dictionary and embedding vector is the output of the trained

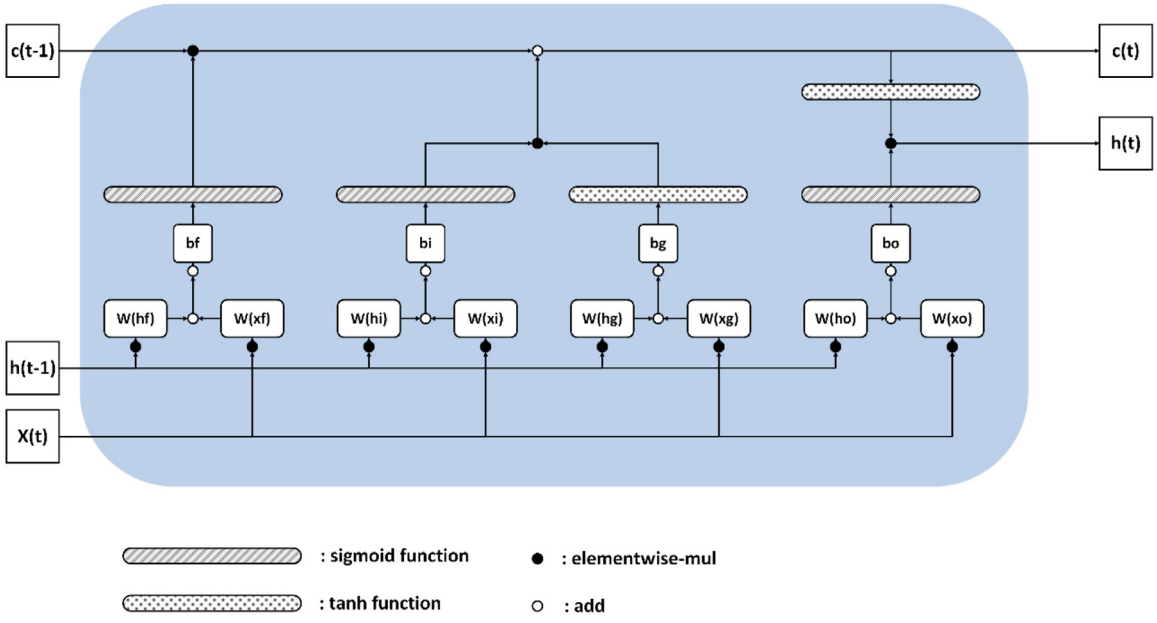

Fig. 4 – LSTM cell.

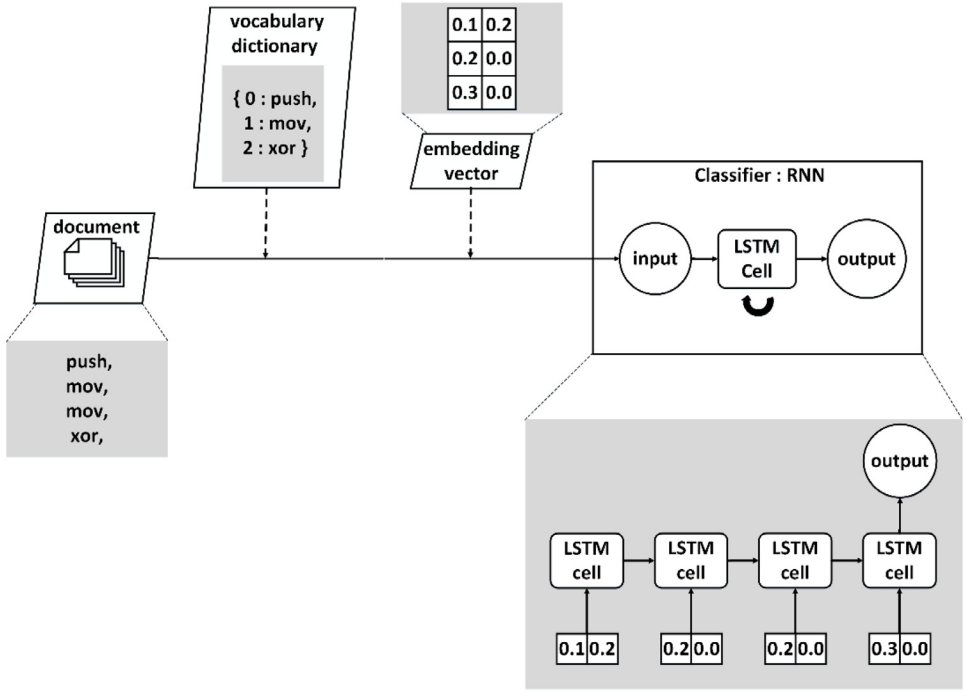

Fig. 5 – The flowchart of classification with an LSTM cell.

Word2Vec model. Then, the embedding vector of each opcode is fed to RNN classifier. The number of opcodes in the document is the value of step size and the dimension of embedding vector is the value of input dimension size in RNN. In the example, the step size is 4 and input dimension is 2. After passing thorough the time step, we only consider the last time step's output as the final output. The target of output is malware or benign. In other words, the classifier learned to classify which document is malware and which is not.

In Damodaran et al. (2017), the occurrences of malware are executed within a lightweight file emulator. The output features obtained from the emulator are then passed to the LSTM network, which is then followed by a "logistic regression classifier".

Similar approach is found in Liu and Wang (2017), in which the supervised approach, like deep learning was used. The use of deep learning has been shown the most effective approach as it accepts raw static and dynamic features without any

expert knowledge. However, their work is significant when large-scale training data is available. Deep learning is a powerful approach to process raw bytes or to recover opcodes by using the techniques of embedded words.

The work in Ye et al. (2018) proposes the same deep learning approach but with an automatic malware detection. But it is based on static analysis in which the malware is termed with the combination of a grey scale image and opcodes.

Assembly opcodes, like jmp, mov, add, sub, etc. were used through different algorithms such as FastText (Athiwaratkun and Stokes, 2017). This algorithm was used specifically for spoken language words. On the other hand, Rhode et al. (2018) compares the result with the same datasets using NVIDIA Quadro P2000 Graphical Processing Unit (GPU) having 1024 CUDA cores and 5 GB of GPU memory. However, the researcher used the Keras deep learning framework. The result shows that it affects the training time and accuracy.

Other researchers also defined and experimented with specific malware language in which the documents were converted into binary forms. These documents were clustered using a tool, namely Word Mover's Distance (WMD) (Bojanowski et al., 2017). But test results were moderate. The same can be seen in (Gibert et al., 2017), in which deep forward neural networks are proposed, but this too shows moderate testing accuracy.

Recent developments in Awad et al. (2018) and Grosse et al. (2016) show an increasing use of Deep Neural Networks for malware detection and subsequent classification. They used RNN to create a few instances to reduce the complexity. Application program interfaced call log sequences are passed to recurrent neural networks (RNN). First five instances of the log files are considered which reduces the complexity of the model. An accuracy of 96% is achieved with the output connected to a Convolutional Neural Network (CNN). However, Tobiyama et al. (2016) starts with a CNN layer. The output is then fed into a recurrent neural network composed of Long Short-Term Memory (LSTM) cells. Max/Mean pooling is applied in order to reduce dimensionality.

2.4. *Proposed method*

This paper proposed a method in which an RNN model is used to effectively detect malware. Firstly, we carefully analyze feature vectors for the texts (documents). For this, we have used three techniques of NLP (Natural Language Processing): (*a*) 'one hot encoding', (*b*) 'random vector representation', and (*c*) 'trained vector representation using a Word2Vec model'. Our method is then followed by several experiments in order to find the influence of hyper parameters on RNN network (e.g. step size, input size, and so on). We performed experiments with RNN with the above-mentioned three feature vectors. We will discuss the performance and stability these, and propose an efficient method with RNN for malware detection.

In Fig. 3, operation steps of neural network in Word2Vec model are represented with an example to show data transformation. Firstly, words of document are converted to one hot encoded vector as (*a*) to (*b*). Then we set (input, target) pairs which are fed to neural network. In this example, we used the Skip-gram model and size of window was 1. Therefore, the pairs can be ('push', 'mov'), ('mov', 'push'), ('mov', 'mov') and so on.

For short, let's just consider the cases of ('mov', 'push') and ('mov', 'mov). Therefore, as can be seen in (*c*), the input vectors are (0, 1, 0), (0, 1, 0) and the corresponding target vectors are (1, 0, 0), (0, 1, 0). After the input vector is fed, the weight between input layer and projection layer do an embedding look up operation. It is fully connected operation that just selects the corresponding vector to a given input. In the example, 'mov' is the second word in the vocabulary. Therefore, after passed through input layer to projection layer, the second row of vector is selected. Then, this embedding vector of 'mov' is trained to close to vector 'push' and 'mov'.

After training is done using these steps, we get an embedding vector matrix which size is number of words in the vocabulary by *n*-dimension. In example (*d*), the words in vocabulary, 'push', 'mov', 'xor' get each embedding vector. Finally, the example document (*a*) can be represented to (*e*).

We know that training on the same malware families will definitely result in exaggerated accuracy. While training the neural network, it required a high computational cost because the need to calculate with whole embedding vector of word in the vocabulary. Therefore, the tricks called hierarchical softmax and negative sampling are applied to Word2Vec (Kaggle, 2014). In this paper, we used negative sampling technique to reduce the computational cost. The idea of negative sampling is calculating loss with sampled negative word, not whole word in the vocabulary. Negative samples are chosen from noise distribution. The negative sampling minimized followed objective function (Gibert, 2016).

$$E = -\log x\left({v'_{wo}}^{T} h\right) - \sum_{w_j \in W_{neg}} \log \sigma\left(-v'^{T}_{wj} h\right) \quad (1)$$

where w_o is the target word and v'_w is its output vector; h is the output value of the hidden layer. $W_{neg} = \{w_j | j = 1, \ldots, K\}$ is the set of words that are sampled based on $P_n(w)$, which is the noise distribution.

After training, the trained embedding vector is the main output of the Word2Vec model. Also, we can get the vocabulary dictionary that represents the word by index.

In this paper, the design of our experiment can be divided into 2 phases. Phase 1 is to create feature vectors, and phase 2 is the classification with RNN classifier. The overall flowchart of experiment is shown in Fig. 6.

In phase 1, we prepared a total of 10,000 datasets, that is, 5,000 malwares and 5,000 benign. The malware data is '.asm' file format that was obtained from the Microsoft malware classification challenge (Kaggle, 2015). The number of each file in experiment is shown in Table 1. Benign data is a PE file format which is randomly collected from local workstations.

We let the test set be 20% of the entire data set and the rest 80% with the training set. The dataset we have used contained 8 different families of malware which are "Ramnit", "Lollipop", "Kelihos_ver3", "Vundo", "Simda", "Tracur, Kelihos_ver1", "Obfuscator ACY" and "Gatak". Each malware files have a 20 characters' hash ID that uniquely identifies the file and class. Microsoft also provided each file with two types of extension which are '.byte' and '.asm'. ".byte" has the file's binary content and is represented in hexadecimal systems.

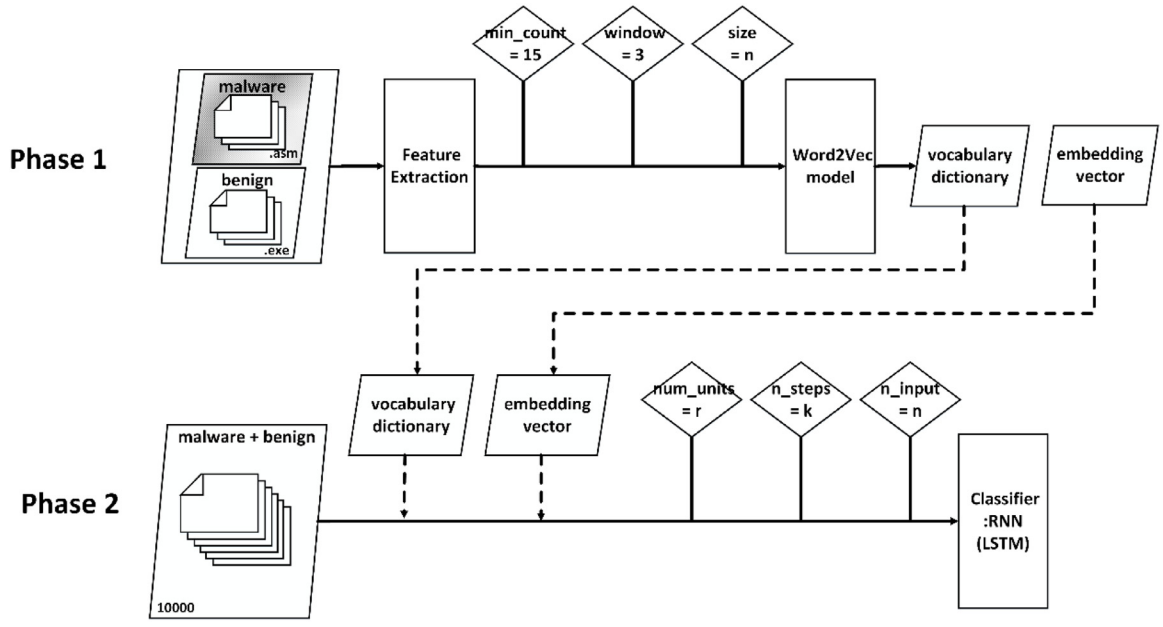

Fig. 6 – The flowchart of the proposed method.

But this extension doesn't have PE header in order to ensure sterility

Then, the prepared data was passed through the feature extraction operation. In the case of one hot encoding, we make feature vectors with only vocabulary dictionary. In the case of random feature vector, we make it with vocabulary dictionary and arbitrary embedding size. In the case of Word2Vec, the extracted feature is passed thorough to Word2Vec model as input. In this paper, the used hyper parameters of Word2Vec model are shown as a diamond shape in Fig. 6. In Fig. 6, the 'min_count' means minimum value of feature frequency to make embedding vectors, the 'window' means the number of considered word to making input and output pair of the model, and the 'size' refers to the dimension of embedding vector.

In phase 2, we feed feature vectors to RNN classifier that used LSTM cell. The hyper parameters of classifier are also shown as diamond shape in Fig. 6. The 'num_units' means the number of neurons in LSTM cell, the '*n*_step' means the step size of RNN, and the '*n*_inputs' means the dimension of input at each step. Also, '*n*_inputs' has the same value as the dimension of embedding vector because each embedding vector is fed into the cell at each step.

We used an RMSProp optimizer with a learning rate 0.001. In the training phase, we used dropout layer with 0.5 probability to avoid overfitting. The whole experiments were processed using Python TensorFlow and genism for the Word2Vec model.

3. Experimental results

In the first step of the second phase, we did experiments with hyper parameters. This is to investigate the effect of hyper parameters. Hyper parameters are variables which determine the network structure and how the network is trained. Hyper parameters are set before training; that is before optimizing the weights and bias.

Firstly, in our experiments, we measured the performance according to different input sizes with fixed step sizes. Values of hyper parameters that are used in this experiment are shown in Table 2.

We did the experiments with different feature vectors using the above hyper parameters. The feature vectors are one hot encoding feature vector, random feature vector and trained feature vector by the Word2Vec Skip-gram model. In the case of one hot encoding feature vector, the input size which is the dimension of each feature, is the same as the size of vocabulary dictionary. This means that all experiments used the same input size. In the case of random feature vector, it is composed of random values according to the truncated normal distribution with a mean value of 0.0 and a standard deviation of 0.1. In the case of Word2Vec feature vector, it is pertained by the Word2Vec Skip-gram model. To measure the performance, we measured the AUC (Area Under the ROC Curve) and accuracy value. It measures a two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). The AUC gives an aggregate measure of performance across all possible classification thresholds. The results are shown in Table 3 and Fig. 7.

Table 2 – The value of Hyper Parameters according to input size.

	HyPa1	HyPa2	HyPa3	HyPa4	HyPa5
Input size (n)	50	100	200	300	500
Step size (k)	500	500	500	500	500
Recurrent Neuron (r)	10	10	10	10	10

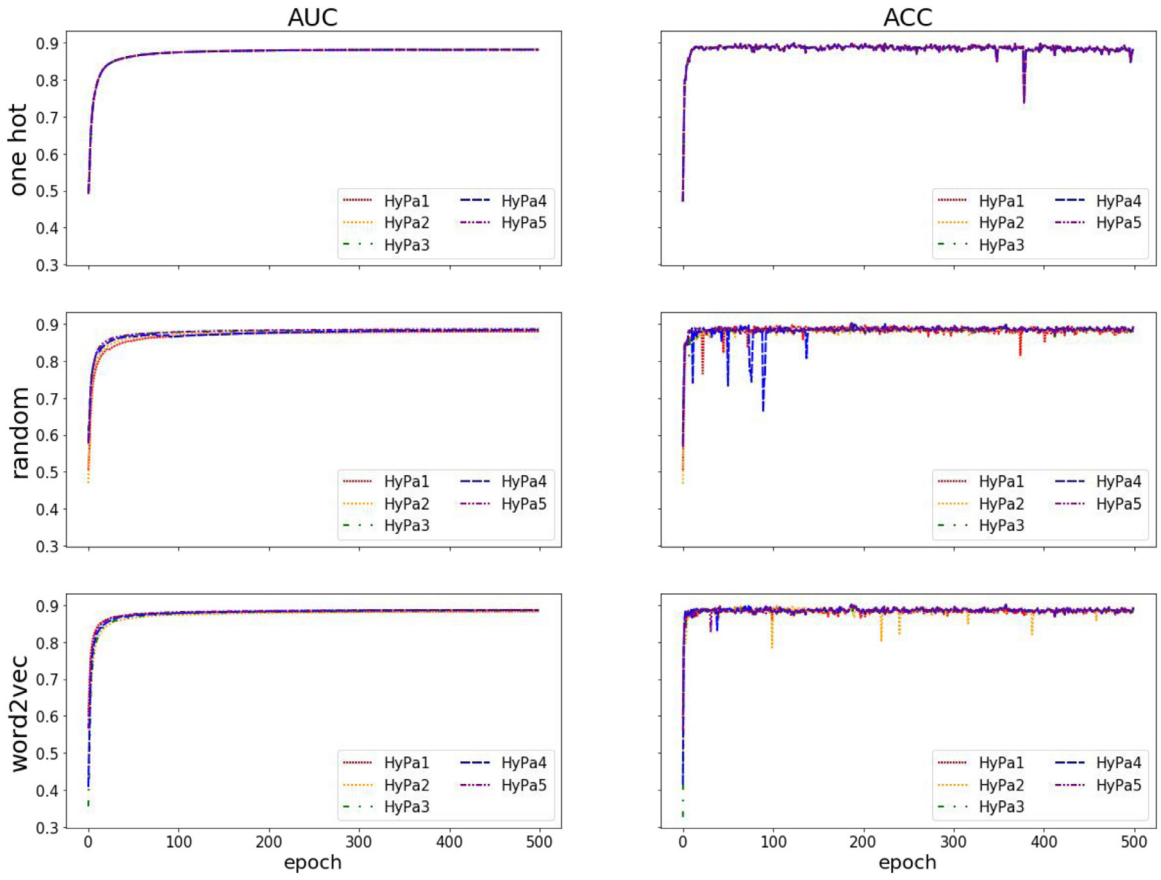

Fig. 7 – The AUC and accuracy (ACC) values according to different input sizes with fixed step sizes.

From results, we can see that the input size did not have a significant effect on performances of RNN. Therefore, we will use a small value of input size (which is 50) for next experiments, as it will be efficient for low memory.

Secondly, we performed experiments with different step sizes with fixed input sizes. The values of hyper parameters that used in this experiment are shown in Table 4.

Table 3 – The AUC and accuracy (ACC) value according to input size with fixed step size.

		HyPa1	HyPa2	HyPa3	HyPa4	HyPa5
One hot encoding	AUC	0.8818	0.8818	0.8818	0.8818	0.8818
	ACC	0.8830	0.8830	0.8830	0.8830	0.8830
Random	AUC	0.8798	0.8828	0.8845	0.8838	0.8866
	ACC	0.8843	0.8930	0.8920	0.8940	0.8943
Word2Vec	AUC	0.8836	0.8834	0.8856	0.8868	0.8873
	ACC	0.8937	0.8937	0.8937	0.8943	0.8947

Table 4 – The value of Hyper Parameters according to step size.

	HyPa6	HyPa7	HyPa8	HyPa9	HyPa10
Input size (n)	50	50	50	50	50
Step size (k)	50	100	200	300	500
Recurrent Neuron (r)	50	50	50	50	50

We used the same feature vectors as the above and the results are shown in Table 5 and Fig. 8.

From results, we can see that the step size did have significant effect on performances of RNN. We found that when the step size becomes bigger, the performance of RNN is getting lower in detecting malware. The main reason is that the dataset is big and is collected from different sources; there is a conflict in the classifiers. Though with dataset problem, we got some insights through these two experiments that step size is more important than input size and the large step size got low performance when using RNN for malware detection.

In the second step of the second phase, we measured the performance according to the feature vectors. This is to investigate the efficiency of feature vectors. In these

Table 5 – The AUC and accuracy (ACC) values according to step size at last epoch in test data.

		HyPa6	HyPa7	HyPa8	HyPa9	HyPa10
One hot encoding	AUC	0.9749	0.9631	0.9154	0.9020	0.8821
	ACC	0.9773	0.9683	0.9233	0.9137	0.8927
Random	AUC	0.9734	0.9638	0.9150	0.9041	0.8800
	ACC	0.9767	0.9680	0.9250	0.9137	0.8923
Word2Vec	AUC	0.9757	0.9653	0.9198	0.9071	0.8820
	ACC	0.9780	0.9680	0.9247	0.9137	0.8943

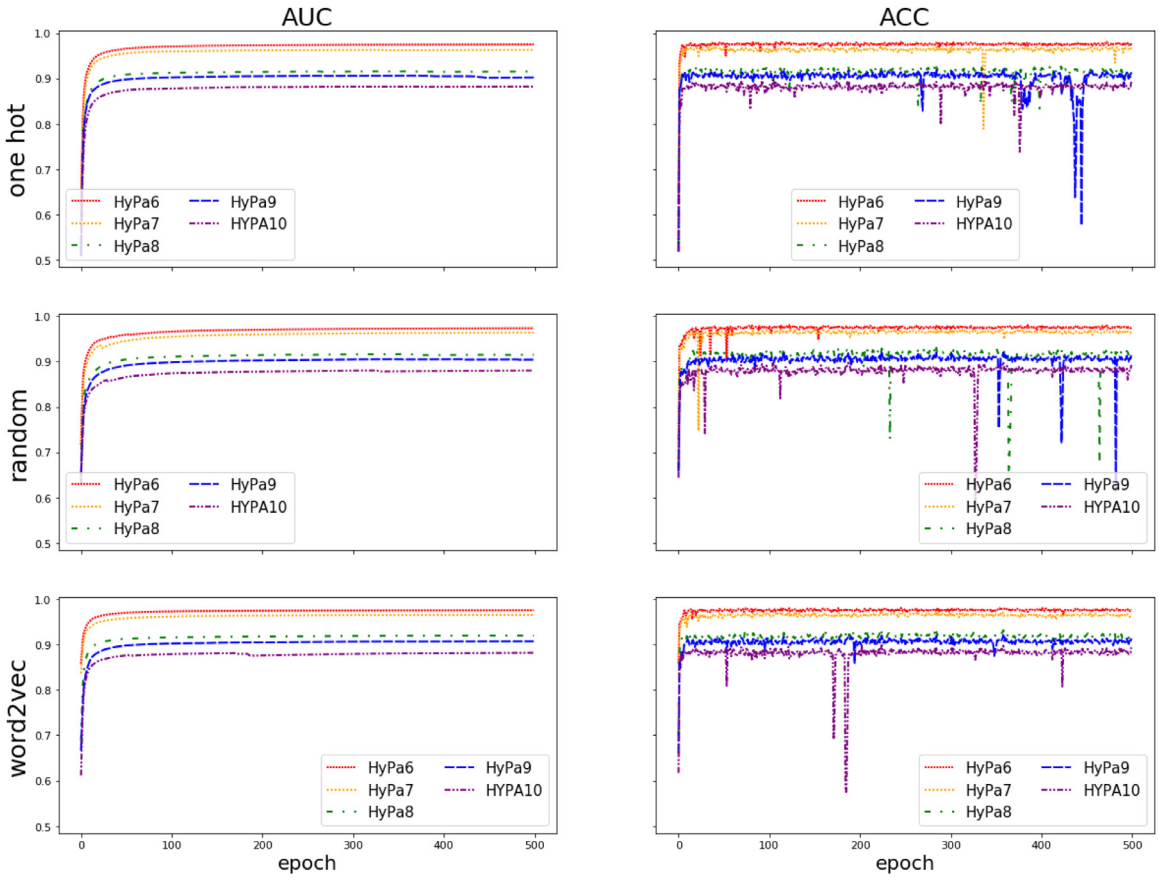

Fig. 8 – The AUC and accuracy value according to epoch in each step size.

experiments, we used the RNN classifier with hyper parameters used in the first step. The input size is 50, the step size is 200, the number of recurrent neurons is 50, and the epoch is 200.

In order to see whether the performance between feature vectors is significant, we did 30 experiments for each feature vector. In the case of one hot encoding feature vectors, they have different vocabulary dictionary. In the case of random feature vectors and Word2Vec feature vectors, they used different random seeds. We measured the performance with the value of AUC, F1 score which is calculated by precision, recall and accuracy. The results are shown in Table 6 and Fig. 9. Fig. 9 shows boxplots of 30 results.

Table 6 – The average (AVG) and standard deviation (STD) values of AUC, F1 score and accuracy (ACC) according to feature vectors.

		One hot encoding	Random	Word2Vec
AUC	AVG	0.9150	0.9122	0.9169
	STD	0.0006	0.0018	0.0012
F1 score	AVG	0.9154	0.9129	0.9176
	STD	0.0007	0.0017	0.0010
ACC	AVG	0.9176	0.9137	0.9191
	STD	0.0029	0.0219	0.0033

We did a paired t-test with a 95% confidence level. We found that the AUC value of each pairs is significant. The result with F1 score was same as AUC value. In the case of accuracy, the random method received the largest variance. Looking at all the results, the Word2Vec feature vector achieved the highest value of average and good variance. Therefore, we recommended that the trained Word2Vec feature vector with small input size using RNN to detect malware.

4. Performance evaluation

From the experimental results, we can see that the input size did not have a significant effect on performances of RNN. The same feature vectors for different step sizes with a fixed input size (which was 50) were used. The results show that when we increase the step size, the performance of RNN is degrading. The reason was that the number of datasets was too high which raised conflicts in the classifier and the larger step size gives a low performance when using RNN for malware detection.

Next, the efficiency of RNN malware detection was evaluated through the feature vectors with input size = 50, step size = 200, number of recurrent neurons = 50 and epoch = 200. The performance measured by AUC and F1 score concluded

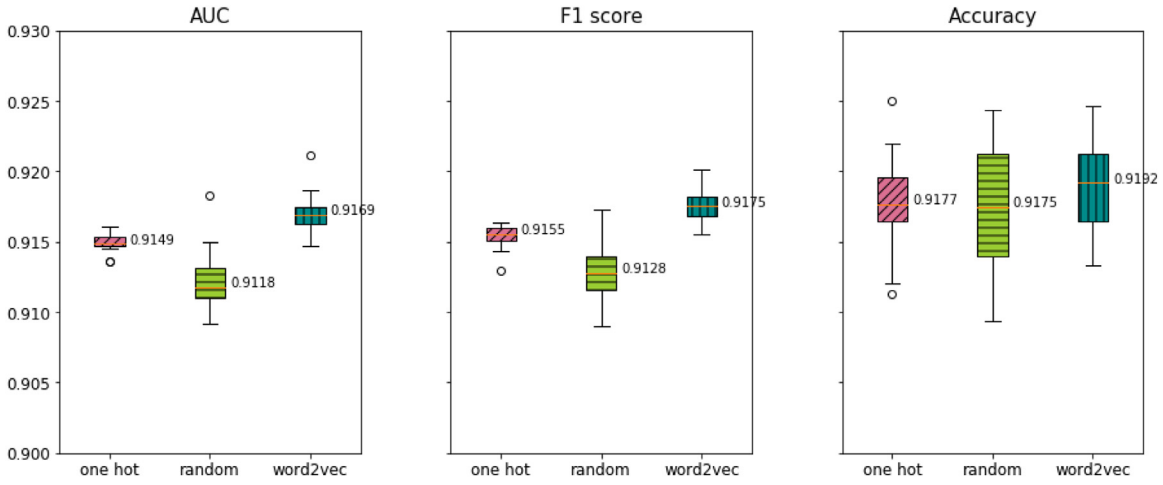

Fig. 9 – The box plot of AUC, F1 score and accuracy according to feature vectors.

Table 7 – Approach based comparative analysis between our model and other models.

Other models	Outcomes/remarks	Outcomes/remarks of our proposed work
Damodaran et al. (2017)	– Tested malware detection techniques using API call sequences and opcode sequences – Trained Hidden Markov Models and compared detection rates for models based on static data, dynamic data, and hybrid approaches	– Tested malware detection using RNN – Classified different malware families – Trained those classified malware families
Andrade et al. (2019)	– Used an approach to identify 5 different types of malware	– Used our approach to identify and test different families of malware
Ahn et al. (2017)	– Proposed malware detection method, which uses feature vectors that consist of Opcode (operation code) and Windows API calls extracted from executable files	– Proposed modified Recurrent Neural Network (RNN) to detect malware through classification of malware families

Table 8 – Result based comparative analysis between our model and other models.

Models	Malicious samples	Benign Samples	Time required to collect data
Damodaran et al. (2017)	745	40	Not explicitly stated
Tian et al. (2010)	1368	465	30 seconds
Tobiyama et al. (2016)	81	69	5 minutes
Ahmed et al. (2009)	416	100	No time mentioned.
Our model	20,000	98 families (each family containing 9 benign samples)	5.3 minutes

that the AUC value of each pair is significant. Our results also show that the Word2Vec feature vector achieved the highest value of average and good variance. Therefore, we recommend that the trained Word2Vec feature vector with small input sizes on applying RNN to malware detection as the best method.

Tables 7 and 8 summarize the differences between our model and other models.

5. Conclusion

In this paper, we proposed an efficient method for malware detection using RNN. We made experiments according to different feature vectors to investigate their efficiency. Those are one hot encoding feature vector, random feature vector, and Word2Vec feature vector. As a result, we found that the trained Word2Vec vector is efficient with RNN for malware classification. Also, we did experiments with different values of RNN hyper parameters to see their effect on them. We found that the step size is more influenced on performance than the input size. Therefore, for memory efficiency, we recommended small input size when using RNN for malware detection. Also, we found that the large step size achieved low performance.

The opcode that is selected as a feature in this paper has some different characteristics from that in general natural languages. Firstly, opcode has far less different kind of words than those in general cases. Secondly, a typical opcode, such as ‘mov’ and ‘add’, is repeated too many times in each file such as ‘mov’ and ‘add’. Thirdly, it is hard to assume that the occurances in similar context opcodes have a

similar semantic meaning. In this point of view, we thought that the pertained feature vector could get similar performances to other untrained feature vectors. In our further work, we will study specialized feature extraction methods for malware detection with reasonable step size and more efficient deep learning model.

6. Limitations and future work

Most of the malware analysis is done by quickly checking the features of a particular piece of code and then examining them. After this, the current malware is compared with the previously found/detected malicious code. However, according to Rhode et al. (2018), behavioral data collected during file execution is more difficult to obfuscate [29]. However, it takes comparatively a long time to identify those malwares. In other words, in most of the cases, the malicious payload gets recovered by the time it is detected. In this paper, we have approached a noble method in which we have trained vector representation to check and validate the influence of hyper parameters, like step size and input size on RNN networks. The results obtained from our experiments indicate a better performance and demonstrates stability in detecting malware. Our method is the first method to predict malware during execution, rather than using a complete activity log file.

Unfortunately, the limitation of our proposed malware detection is the overall computational complexity over two or more large datasets. The presented work in this paper has some limitations, which include different characteristics of the opcodes. When we consider multiple datasets of large data volume, the repetition and occurrence of similar context opcodes result into the same semantic meaning. Further, the increase in the step size reduced the performance due to the high number of datasets; which resulted in the conflict of classifiers.

As our further work, we will study the specialized feature extraction methods for malware detection with reasonable step size and more efficient deep learning model. For large datasets that contain malicious software, classification techniques become imprecise. This hinders the issues of detecting malwares using RNN and we have also noted this as our future work.

Declaration of Competing Interest

The authors declare that: They have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper; They have no financial interests/personal relationships which may be considered as potential competing interests. This statement is agreed by all the authors to indicate agreement that the above information is true and correct.

CRediT authorship contribution statement

Sudan Jha: Supervision, Conceptualization, Methodology, Investigation. **Deepak Prashar:** Data curation, Software, Writing - original draft. **Hoang Viet Long:** Supervision, Validation. **David Taniar:** Supervision, Validation.

REFERENCES

Ahmed F, Hameed H, Shafiq MZ, Farooq M. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In: Proceedings of the 2nd ACM workshop on security and artificial intelligence. ACM; 2009. p. 55–62.

Ahn TH, Oh SJ, Kwon YM. Malware detection method using opcode and windows API calls. J. Inst. Internet, Broadcast. Commun. 2017;17(6):11–17.

Andrade EO, Viterbo J, Vasconcelos CN, Guérin J. Bernardinia FC. A model based on LSTM neural networks to identify five different types of malware. Procedia Comput. Sci. 2019;159:182–91.

Athiwaratkun B, Stokes JW. Malware classification with LSTM and GRU language models and a character-level CNN. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); 2017. p. 2482–6. doi:10.1109/ICASSP.2017.7952603.

Awad Y, Nassar M, Safa H. Modeling malware as a language. In: Proceedings of the IEEE International Conference on Communications (ICC); 2018. p. 1–6.

Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. Trans. Assoc. Comput. Linguist. 2017;5:135–46.

Damodaran A, Troia FD, Visaggio CA, Austin TH, Stamp M. A comparison of static, dynamic, and hybrid analysis for malware detection. J. Comput. Virol. Hack Technol. 2017;13:1–12.

Géron A. Hands-on Machine Learning with Scikit-Learn & TensorFlow, First ed. O'Reilly Media 2017:379–403.

Gibert D, Bejar J, Mateu C, Planes J, Solis D, Vicens R. Convolutional neural networks for classification of malware assembly code. In: Proceedings of the 20th International Conference of the Catalan Association for Artificial Intelligence; 2017. p. 221–6. http://ebooks.iospress.nl/publication/47742.

Gibert D. Convolutional Neural Networks for Malware Classification. A thesis presented for the degree of Master in Artificial Intelligence. Escola Politècnica de Catalunya (UPC) - BarcelonaTech, Universitat de Barcelona (UB), and Universitat Rovira i Virgili (URV); 2016.

Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435, 2016.

Hiai S, Shimada K. Sarcasm Detection Using RNN with Relation Vector. Int. J. Data Warehousing Min. 2019;15(4):13.

Kaggle. Microsoft malware classification challenge, https://www.kaggle.com/c/malware-classification, 2015, (accessed 15 January 2019).

Kim Y. Convolutional Neural Networks for Sentence Classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); 2014. p. 1746–51. doi:10.3115/v1/d14-1181.

Long HV, Tuan TA, Taniar D, Can NV, Hue HM, Son NTK. An efficient algorithm and tool for detecting dangerous website vulnerabilities. Int. J. Web Grid Serv. 2020;16(1):81–104.

Liu L, Wang B. Automatic malware detection using deep learning based on static analysis. In: Proceedings of the Third International Conference of Pioneering Computer Scientists, Engineers and Educators, ICPCSEE; 2017. p. 500–7.

Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. Proceedings of International Conference on Learning Representations (ICLR), 2013.
Popov I. Malware detection using machine learning based on Word2Vec embeddings of machine code instructions. Proceedings of the Siberian Symposium on Data Science and Engineering (SSDSE), 2017.
Ranveer S, Hiray S. Comparative analysis of feature extraction methods of malware detection. Int. J. Comput. Appl. 2015;120(5):1–7.
Rhode M, Burnap P, Jones K. Early-stage malware prediction using recurrent neural networks. Comput. Secur. 2018;77:578–94.
Rong X. Word2Vec Parameter Learning Explained. CoRR abs/1411.2738, 2014.
Schultz MG, Eskin E, Zadok E, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings of the IEEE Symposium on Security and Privacy; 2001. p. 38–49.
Siddiqui M, Wang MC, Lee J. Data mining methods for malware detection using instruction sequences. In: Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications; 2008. p. 358–63.
Tian R, Islam R, Batten L, Versteeg S. Differentiating malware from cleanware using behavioural analysis 2010 5th international conference on malicious and unwanted software; 2010. p. 23–30.
Tobiyama S, Yamaguchi Y, Shimada H, Ikuse T, Yagi T. Malware detection with deep neural network using process behavior, 2; 2016. p. 577–82. doi:10.1109/COMPSAC.2016.151.
Veeramani R, Rai N. Windows API based malware detection and framework analysis. Int. J. Sci. Eng. Res. 2012;3(3):1–6.
Wetzker R, Zimmermann C, Bauckhage C. Detecting trends in social bookmarking systems: a del.icio.us endeavor. Int. J. Data Warehousing Min. 2010;6(1) 20 pages.
Ye Y, Chen L, Hou S, Hardy W, Li X. DeepAM: a heterogeneous deep learning framework for intelligent malware detection. Knowl. Inf. Syst. 2018;54:265–85.

Hoang Viet Long is currently working as the researcher of Institute for Computational Science at Ton Duc Thang University, Ho Chi Minh City, Vietnam. He obtained Ph.D. diploma in Computer Science at Hanoi University of Science and Technology in 2011, where he defensed his thesis in fuzzy and soft computing field with applications to electronic engineering. He has been promoted to Associate Professor in Information Technology since 2017. Recently, he has been concerning in Cybersecurity, Machine Learning, Bitcoin and BlockChain and published more than 40 papers in ISI-covered journal.

Deepak Prashar received the B.Tech in Information Technology from uttar pradesh Technical University, UP, India. he has completed his M.Tech. in Information Technology and Communication (ICT) from the School of Information Technology and Communication, GBU Greater Noida, India in 2012 and currently doing Ph.D. in Computer Science from the School of Computer and Systems Sciences, JNU, New Delhi, India. His primary research interests are Wireless Sensor Body area Network and Network Security.

Sudan Jha was born in Kathmandu, Nepal. He received proficiency in certificate level from the Saint Xavier's College, Kathmandu and the B.E. degree in electronics engineering from the Motilal Nehru Regional College, Allahabad, Uttar Pradesh, India, in 2001. He joined as a Lecturer with the Nepal Engineering College (nec), one of the premium and largest engineering college and the first one in the private domain in Nepal, where he got full sponsorship from the employer (nec) to pursue master's degree in computer science. He was an Assistant Professor with the Department of Computer Science and Engineering after completion of his master's degree and no sooner than later, he became the Head of the Computer Science and Engineering Department, Nepal Engineering College. In due course of time, he chaired and organized five international conferences, and some of the proceedings of those conferences had been published by Springer Verlag, World Science Series, and Imperial Press London.

David Taniar received the bachelor's, master's, and Ph.D. degrees all in computer science, specializing in databases. His research areas are in big data processing, data warehousing, and mobile and spatial query processing. He has published a book on the High-Performance Parallel Database Processing (Wiley, 2008). He has also published over 200 journal papers. He is a regular keynote speaker at an international conference, delivering lectures and speeches on big data. He is a founding Editor-in-Chief of the International Journal of Web and Grid Services, and the International Journal of Data Warehousing and Mining. He is currently an Associate Professor with Monash University, Australia.