Int. J. Advance Soft Compu. Appl, Vol.11, No. 1, March 2019
ISSN 2074-8523

Recurrent Neural Network for Malware Detection

Mudzfirah Abdul Halim[1], Azizi Abdullah[2], Khairul Akram Zainol Ariffin[3]

[1]Faculty of Information Science and Technology (FTSM),
[2]Center for Artificial Intelligence and Technology (CAIT),
[3]Center for Cyber Security (Cyber),
Universiti Kebangsaan Malaysia, 43600 UKM, Bangi Selangor, Malaysia
e-mail: mudzfirahalim@gmail.com, {azizia, k.akram }@ukm.edu.my

Abstract

Recently, an active development of network communication technology has brought inspiration to new cyber-attack such as malware. This possesses a massive threat to network organization, users and security. Consequently, many researchers have developed novel algorithms for attack detection. Nevertheless, they still face the problem of building reliable and accurate models that are capable in handling large quantities of data with changing patterns. The most common technique to represent the feature of malware is bag-of-words (BOW) where the frequency of each word is used for malware description. However, using BOW approach will destroy the spatial and sequence information aspects of malware patterns, resulting in information loss and coarse indexing. Therefore, this paper presents two combination models of Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) to deal with spatial and temporal signals problem of BOW representation. Both techniques are well known in the classification problem with LSTM being useful in temporal modeling while CNN is good at extract spatial information from data. After that, the Multi-Layer Perceptron (MLP) is used for classification. The model is trained on Drebin dataset and validated, and then the result is compared with other techniques. The experiment shows that the both proposed models outperform common MLP, CNN and LSTM models on a malware classification task. Our best model (LSTM-CNN) model obtains state-of-the-art performance level of 98.53% of the Drebin dataset.

Keywords: *Deep Learning, Long Short Term Memory, Malware Classification, Recurrent Neural Network.*

1 Introduction

In this century, the development of network technology changes people life where they could easily access information from around the world more effectively than previous years. At the same time, however, a great number of cyber incidents such as malware also actively evolving. Malicious software, commonly known as malware is a continuous problem and has become a major threat to computer users, businesses, corporations, and even governments. The number of malware increasing year by year and becomes more complex and sophisticated. It is harmful and can contribute to unwanted loss or privacy invasion as it compromises the confidentiality, integrity and availability of private data without user's permission. Consequently with the rising of cyber incidents related to malwares, most researchers have studied various techniques to detect them.

Machine learning techniques such as Support Vector Machines (SVM), k-Nearest Neighbor (k-NN), Naïve-Bayesian (NB), Random Forests (RF), Neural Network (NN), are prominently explored for malware classification[1]–[4]. Several researchers combine numerous machine learning classifiers as they claim the hybrid of multiple classifiers [5] has better performance than single classifier. Andrew H. Sung combined SVM with Artificial Neural Network (ANN) in his work and reduced the number of features for better performance but the result was undifferentiated to the result of original features [6]. Same claim from Maheshkumar Subhnani et al. who combined Multilayer Perception (MLP), k-Means and decision tree in his paper [7] to detect malware, also showed that the performance improved in detection and false alarm rates. Advance machine learning has also been applied for malware detection since it is very convenient in extracting more information from the datasets. Even though unsupervised machine learning such as clustering task seems to be the most preferable by researchers to understand malware [8][9], the supervised machine learning method is also preferable when it comes to correct labeling [10]. Dahl et al. [11] highlighted the use of supervised machine learning to classify the labeled sample of malware by combining random projection and NN techniques.

In the past few years, there were noticeable work involving Deep Neural Networks (DNN) in classifying malware [1], [11]–[13]. Saxe et al. [1] utilized feed forward NN for static analysis. However, as the focus was on static analysis and dealing with binaries of executable files, the satisfactory input for the classification was not achieved. Motivated by Pascanu et al. [12] that learnt malware through language model, Athiwaratkun et al. [14] and Kolosnjaji et al. [13] expanded the research by using Recurrent Neural Network (RNN) to enhance malware sequence classification. In addition, a combination of RNN with MLP was applied in [12] to learn malware and benign files through language model and formed feature representation. In this work, the MLP was allocated as output classifier while RNN worked as feature extractor. The usage of temporal max pooling helped improved and produced the best result in processing long sequence of temporal features but

RNN failed to learn the outstanding features of malware. Furthermore, RNN produced a lower detection rate in almost every experiment, compared to Echo State Network (ESN). Athiwaratkun & Strokes achieved 31.30% improvement in detection rate by combining Long Short Term Memory (LSTM) with Gated Recurrent Unit (GRU) as a language model [14].

Nevertheless, input for machine learning and data science need to be represented numerically and the popular technique that is commonly used to convert word input into vector input is Bag of Word (BOW) model. This model represents the frequency of each word as feature sequence input. However, one of the main technical issues addressed in using BOW representation is the spatial and temporal problem against the sequences input especially when multi-step predictions have to be made [15]. This is because BOW model destroys the spatial relationship of feature sequences which limit its descriptive ability. Spatial data is important as it provides the information linkage between features and it is helpful in terms of better understanding of the way the feature sequences of malware are related to each other.

LSTM comes with memory cells that can solve temporal problem by learning the temporal structure between each sequential input and then achieve a high level abstraction of data because of its complex architecture. Kim et al. [16] applied LSTM for intrusion detection on KDD Cup 1999 dataset and achieved a higher detection rate compared to other machine learning classifier. Combination of classifiers applied in [15] to deal with temporal data where CNN was used in the first layer to reduce variance in the frequency of input and then the output was fed to LSTM for temporal modeling. Therefore, in this paper, LSTM and CNN are combined to detect malware more accurately and then the accuracy is compared with other DNN classifiers. The idea of combining DNN especially LSTM and CNN for classification has been explored before, though previous work was in different domain field. However, layering order of algorithm plays big role in producing the best performance in malware detection when two different algorithms are combined. Hence, the Drebin dataset is utilized which is one of the few public malware data that represents features with words. The words are then transformed into feature vector dataset using BOW model. This dataset is formulated as temporal sequence malware problem that can be solved by passing the sequence feature input through LSTM for temporal modeling and fed into CNN layer for feature pattern analysis. Lastly, output features are classified by MLP classifier. It is believed that malware classification can be improved by combining these NN as a model. This paper is organized in five sections; Section 1 is the introduction, Section 2 reviews related work in the area, section 3 presents the proposed malware detection model while section 4 discusses experimental results and comparative analysis. Finally which is Section 5, concludes the paper with summary and future work directions.

2 Background

In this section, we will briefly explain the fundamental of Neural Network models.

2.1 Artificial Neural Network

Artificial NN (ANN) mimics the processes happening in real human neurons. The neurons in brain communicate with each other by sending electrical pulses through wiring called synapse. The input signal is first received via dendrites and after that it is processed by soma cell. The cell turns the processed value into output via axons and synapses. Fig. 1 illustrates the basic architecture of ANN.

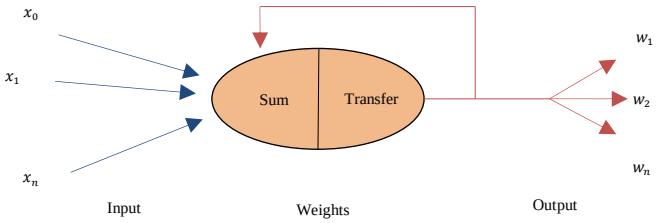

Fig. 1 The basic architecture of ANN

x_n and w_n represent the input and weight of input respectively. ANN model receives input x_n, and next they are weighted with value w_n based on their importance and later all the input are added. The sum value is then fed through transfer function and output is generated. The output is determined by the weighted sum of input and weight, $\sum_i w_i x_i$. To be precise, the equation (1) is shown as below:

$$\text{Output}= \begin{cases} 0 \; if \; \sum_i w_i x_i \leq threshold \\ 1 \; if \; \sum_i w_i x_i \; > threshold \end{cases} \qquad (1)$$

2.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a practical technique in classifying sequences. There are a number of tasks that include the RNN in their operation such as image captioning, speech recognition, sentiment analysis and scene labelling. RNN is an extension of regular ANN with the purpose to enhance performance. The ANN possesses a few drawbacks such as inability to deal with temporal data which requires fix input and output size. This is because ANN is independent and omits everything from previous feed-forward input. It concentrates only on particular input and then maps them directly to output vector. Output from sequential input is only significant when all inputs are dependent on each other because the whole input is useful. In contrast, the RNN offers more flexibility in processing various sizes of input and output using its memory that allows it to produce output that function dependently based on the entire history of input.

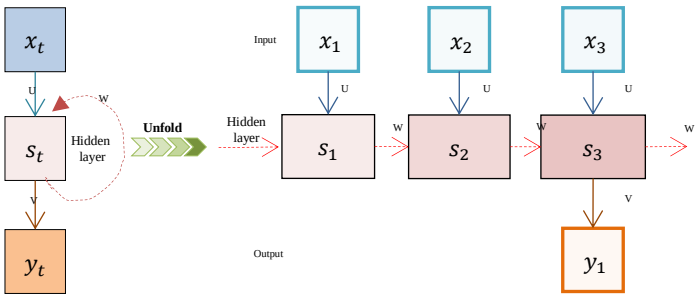

Fig. 2 Traditional RNN

The layer in RNN consists of input layer, hidden layer and output layer. Fig. 2 gives a simple example of RNN that consists of many input unit, one output unit and hidden layer. The x_t represents input, while the y_t is an output and s_t is the hidden state value. All these variables are measured at time step. The hidden state is known as the memory; computed at a particular input and is carried forward by the network. By tracing the arrow coming towards hidden state value, there are two variables to compute new hidden state which are input and previous hidden state value. U, V, and W refer to the parameters for the different layers. In ANN, these parameters varied at each layer but in RNN, the same parameter values are used for each layer throughout to the end.

The hidden state value (2) and output (3) are calculated as follow:

$$s_t = \sigma(w_{xs}x_t + w_{ss}s_t + b_s) \quad (2)$$

$$y_t = w_{sy}s_t + b_y \quad (3)$$

In the above formula, σ function is nonlinearity, w is a weight matrix, and b is a bias form. There are different types of weight matrices and each matrix has different explanation. w_{xs} maps the input value x to hidden state value s. The w_{ss} maps the value of hidden state s to another hidden state value along the time axis. For instance from s_1 to s_2, w_{sy} maps the hidden state value to an output value y. There are also constant biases in RNN which are denoted as b_s and b_y for hidden state and output respectively. This bias vector can vertically shift any value passing through the activation function. However, RNN take advantages of backpropagation called Backpropagation Through Time (BPTT). It computes the gradient across the many time steps [17] and faces a significant gradient vanishing problem as mentioned in [18].

2.3 Long Short Term Memory

Long Short Term Memory (LSTM) [19] proposed to solve the vanishing gradient problem addressed by RNNs. The architecture of LSTM is slightly different when compared to RNN due to the existence of complicated mechanism named memory cell. The memory cell learns the input in an intelligent way to enable the LSTM network to process and store the information for short-term as well as long-term memory. Fig. 3 illustrates a typical structure of memory cell. It receives three types of inputs which are c_{t-1}, s_{t-1} and x_t that represents the cell state from previous cell memory, the previous hidden state value and input respectively. These inputs compute new cell state denoted by $c_t$which then compute the new hidden state denoted by s_t.

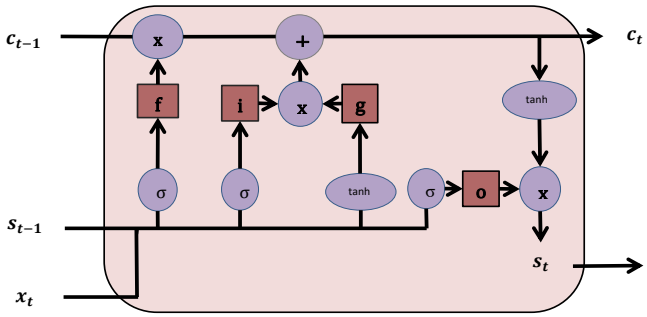

Fig. 3 Memory cell of LSTM

Gating concept in LSTM controls the information entering and leaving the memory cell. The gates f, g, i and o are the forget gate, write gate, input gate and output gate respectively. Forget gate resets the information such as hidden state value from previous cell state which is no more used in current memory cell. Write gate is a gate that scans and chooses the value to be added to the new information in the cell state. The input gate is responsible to write a process where it decides the number of information needed by gate g to be added into the cell state. The output gate reads the cell state from memory and produces a hidden state vector. All computations for the gates are as follow:

$$f = \sigma(\mathrm{w}_{xf}x_t + \mathrm{w}_{sf}s_{t-1} + \mathrm{w}_{cf}c_{t-1} + b_f) \quad (4)$$

$$g = \tanh(\mathrm{w}_{xg}x_t + \mathrm{w}_{sg}s_{t-1} + b_g) \quad (5)$$

$$i = \sigma(\mathrm{w}_{xi}x_t + \mathrm{w}_{si}s_{t-1} + \mathrm{w}_{ci}c_{t-1} + b_i \quad (6)$$

$$o = \sigma(\mathrm{w}_{xo}x_t + \mathrm{w}_{so}s_{t-1} + \mathrm{w}_{co}c_t + b_o) \quad (7)$$

The σ function is the activation function, which stands for standard sigmoid function. This function lies between zero and one. A zero value function will not let any information pass through while a value of one means letting all information enter the gate. Similar to hidden state in RNN, each weight in LSTM is also descriptive; each weight represents the weight matrices that connect the peephole between one information to another exposing the internal state. For instance in

equation (4), w_{xf} represents the weight that map input x to forget gate f. $w_{sf}s_{t-1}$ maps hidden state value from previous cell to forget gate while $w_{cf}c_{t-1}$ maps value of memory cell in previous memory.

These gates play big role in information control. Equation (8) shows the procedure of updating the new value of cell state by forgetting the previous cell value using gate f and inputting the current cell value through gate i and g.

$$c_t = f * c_{t-1} + i * g \tag{8}$$

The activation function of new cell value will be processed through gate o, and new amount of output will be produced which is known as the hidden state value. This process can be formulated as in equation (9):

$$s_t = o * \tanh(c_t) \tag{9}$$

2.4 Convolutional Neural Network

Convolutional Neural Network (CNN) [20] was designed initially for image recognition.Fig. 4 illustrates simple CNN that consists of input layer, convolution layer, max-pooling layer and output layer.

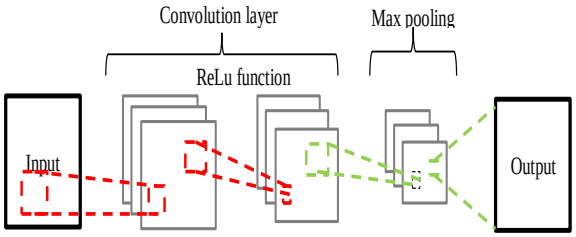

Fig. 4 Convolutional Neural Network

CNN model has the ability to recognize local features between inputs that enable this model to learn features regardless of the position they are placed in the input matrix. The convolution layer will extract the features' input and learn the pattern by applying the convolving filters to the features' input. The output is generated by multiplying local input by the filters weight value. This convolution process represents the characteristics of CNN where it allows the replication of input to be done by sharing the same filter unit. The convoluted output will be fed into max-pooling layer where the variability removal is carried out. This input reduction process involves comparison between the convoluted output and maximum value that is chosen. It partitions the convoluted output into a set of windows that searches for the maximum values by comparing each values. This operation plays big role to help eliminate smaller values and provide invariant output.

3 Malware Detection Model

In this section, the malware detection model for solving the spatial and temporal problem is presented. Instead of using raw Drebin dataset, malware features with the application of BOW representation are treated as malware classification problem. The input of the model is the malware features, and the output is a binary classification.

3.1 Malware features

The Drebin dataset used to train and validate proposed models. In total, this dataset contains 129,013 samples of real world application with features. The details of the malware features were as listed in Table 1.

Table 1 Details of features

Feature name	Description
Hardware components	This feature represents the requested hardware. Permission to certain hardware component has few security issues as it can reveal private data.
Requested permissions	Permission of a system is one of the important security mechanisms in an Android. A malicious application tends to request for permission more often to grant access into the system compared to normal application.
Application components	This feature declares suspicious component that exists in application such as activities and services provided. The declaration helps in analysing and identifying malware as it usually shares the same particular name of services.
Filtered intents	Intents collect all the communication and information during the inter-process and intra-process in Android. This feature is important as malware tends to follow specific intents.
Restricted API calls	This feature lists the critical API calls that are restricted by Android permission system. These API calls can give a deeper understanding on the function of an application.
Used permissions	The restricted API calls will be analysed to determine which requested permission is actually used. This feature lists the permissions that are requested and eventually used.
Suspicious API calls	Certain suspicious calls that are frequently requested by the malware.
Network addresses	Malware needs to establish a network connection to collect data from a device. This feature lists all the addresses that are used by the malware.

Drebin dataset does not provide numerical value for both malware and benign applications, so the pre-processing was needed. To make them available for our model to train, we first detect the features using common tokenizer. Tokenizer splits text into individual word. The first word for every single line in the sample were chosen and listed. Based on the features' list, it was found that few samples had sets of feature; services providers and services receivers. Service provider is a feature that declares services provider of application while service receiver feature declares the service receivers of application. Service is a component that runs in the background until it stop itself and does not interact with user directly. Malwares are likely request these two feature more frequent than benign application. Thus, these two had been chosen as additional features for malware classification model evaluation. For feature extraction, Bag of Words (BOW) model was applied. In this BOW model, specific list of words were retrieved and unimportant words were removed from the document. It helps document retrieval by matching the chosen words in the list and counts their frequency. All gathered features were stored in .txt file for further analysis. Fig. 5 presents pseudocode for BOW model in transforming word sequences to numerical sequences. The model read every line in the documents and ignored characters and words that were not in the feature dictionary. Words that matched the list of features were retrieved and frequency of its occurrence was counted.

```
List_feature = [Hardware components, Requested permissions, Application components,
                Filtered intents, Restricted API calls, Used permissions, Suspicious API calls,
                Network addresses, Services provider, Services receiver]
For each line in dataset:
        If line!=null then do split word by placing a space before and after characters ::
        (this is to ensure that only selected words in DREBIN dataset are extracted as
        feature vector);
        Read every line
        Tokenize the word by splitting it on spaces
        Remove tokens of space, empty string or punctuation marks
For each tokenized word:
        Set id
        If tokenized word=List_feature[]:
        Freq[id]++;
```

Fig. 5 Pseudocode of BOW model

In addition, due to uneven number of samples between malware and benign, new training and test dataset were generated evenly. 2779 random samples from benign and 2779 from malware were chosen. Then, all of the samples were combined in new dataset.

3.2 CNN-LSTM

Fig. 6 shows the overview of detection model CNN-LSTM where CNN stacked on top of LSTM. The idea behind this combination model is that the feature filtration will be done first before the temporal modelling performed. The dataset passed through CNN at the first layer for feature reduction and next the result of smaller feature dimension fed into LSTM layer for data sequences learning. Lastly, output features from LSTM are classified by MLP classifier.

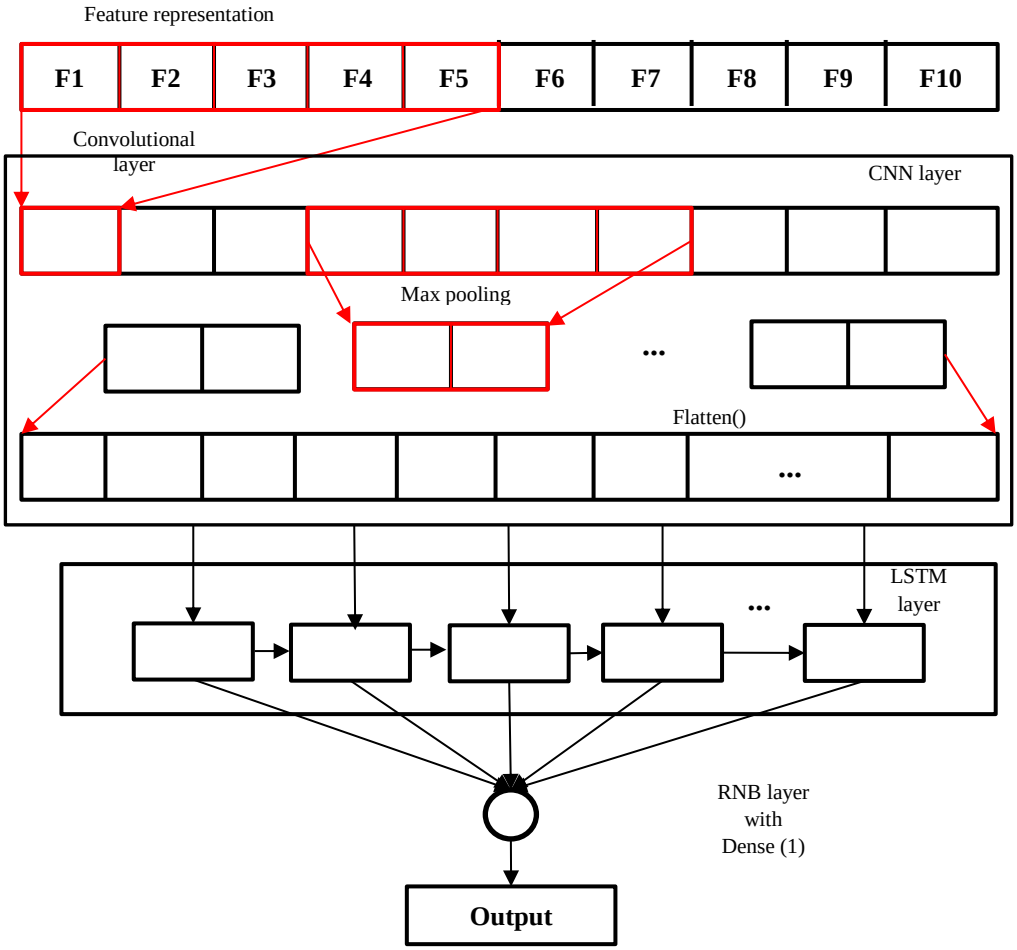

Fig. 6 CNN-LSTM detection model

3.3 LSTM-CNN

Fig. 7 shows the overview of the detection model LSTM-CNN. LSTM is put as the first layer to learn temporal data from BOW representation data. The architecture can be seen as a deep architecture through time steps with LSTM memory cells to produce output sequences. This model learns end-to-end features from malware

feature sequences where it is extracted layer by layer. The temporal modeling using LSTM is performed. One layer of LSTM is used with memory cells to remember all feature inputs. The LSTM output is then passed to the CNN to reduce the feature variation. The architecture used is one dimension convolution layer, with 5x1 feature filters shared across the space. A 4x1 size of max-pooling is then performed on the convoluted output.

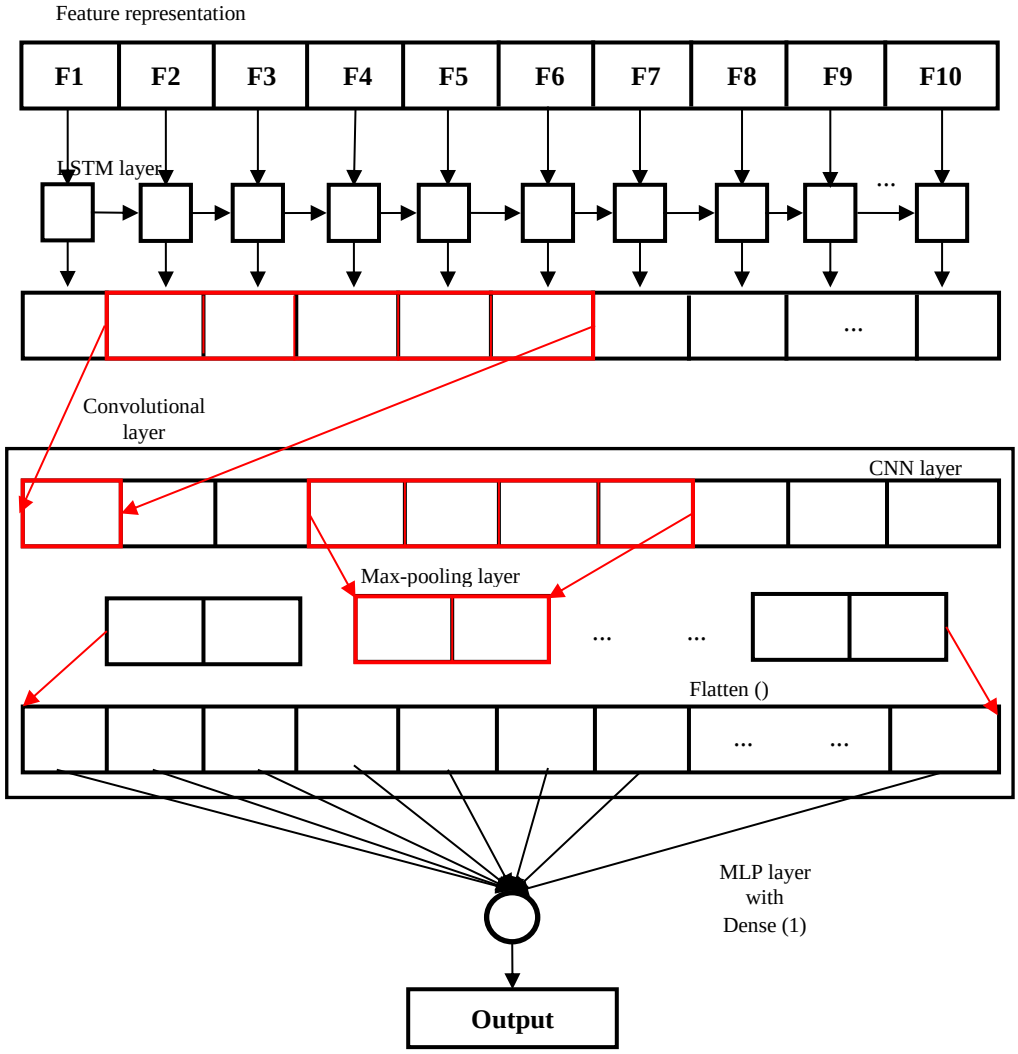

Fig. 7 LSTM-CNN detection model

The MLP layer is added after the output of CNN for classification task. In this layer, all outputs are stacked and classified using single sigmoid function. It then sends the classified output for comparison to evaluator which receives two set of

datasets; one from MLP and another one from feature dataset for validation. Evaluator fully utilize dataset from MLP classifier that contains an output that is produced based on learning algorithm and validate dataset from feature dataset to determine the maliciousness of malware. The decision is then compared with the label provided to check the accuracy of the model.

3.4 Training

Before designing the model development, the malware detector model is optimized on computer with Windows 7 environment using Intel(R) Core(TM) i3-2350M 2.30GHz and 6GB RAM. The model shares almost the same optimization parameters. Parameter gives impact to the performance of model. There are parameters values that can be tuned such as batch size, optimizer, learning rate, number of epochs, number and size of layer and activation function. However, among all the parameters, researcher found that the learning rate (lr) of the optimizer and size of neuron have the greatest impact. Therefore, the parameter experiment is run by changing the values of lr, number of neuron and epoch. Adam optimizer is used because it computes individual adaptive learning for different parameters. Adam maintained the learning rate for each parameter and separately adapts as learning unfolds. As a result, the best values for lr, number of neuron and number of epoch for LSTM are 0.01, 90 and 60 respectively.

4 Experiment

This section presents the experimental result and evaluation of the experiment on Drebin dataset. Before conducting the experiment, the dataset was transformed into vector using BOW model. It contained 10 feature vectors with label 0 for benign and 1 for malware. For the hidden layer, LSTM and CNN architecture were applied; and sigmoid function of MLP was applied to the output layer

4.1 Drebin dataset

In this paper, the Drebin dataset [21] was utilized to evaluate the performance of the proposed model. Since a lot of researches have applied this dataset, it is fair that it was selected for benchmarking purpose. In general, this dataset contains 129,013 samples with eight features which were hardware component, requested permissions, application components, filtered intents, restricted API calls, used permissions, suspicious API calls and network addresses. 123,453 samples of this dataset represent the benign and the rest were malware. Each of malware samples belongs to one of 179 malware families. In our experiments, we used the frequency of feature occurrence in the application text file. For example, a sample of application can have feature vector of 23, 89, 90, 56, 78, 1, 0, and 22. This means the first feature occurs 23 times and so on.

4.2 Evaluation Measures

A In order to evaluate the detection performance, evaluation metrics that were derived from the confusion matrix were used. The following measures were derived and applied in evaluator:

$$TPR = TP/ (TP+FN) \quad (9)$$

$$FPR= FP/ (TN+FP) \quad (10)$$

$$Accuracy = (TP+TN)/ (TP+TN+FP+FN) \quad (11)$$

TPR or True Positive Rate is the ratio of malware detected by the model. FPR or false-positive rate is the value of benign data that is incorrectly classified as malware. The accuracy metric is the rate of performance in detecting malware. TP or true positive is the number of benign samples that is correctly classified by the model. TN, or true negative, is the number of malware samples that is correctly classified. FP, or false positive, is the number of benign samples that is wrongly classified as malware and FN, or false negative, is the number of malware samples that is wrongly classified as benign.

4.3 Measuring Performance

In terms of evaluation, dataset were partitioned to 10 sub dataset. In each dataset, there were 5558 randomly selected samples that consist of equal proportion for both malware and benign. Both training and testing dataset were randomly selected and labelled. 10 times evaluation experiment was run for each dataset using the optimized parameter to train the model and the result was recorded. The optimized parameter for each model was summarized in Table 2.

Table 2 Comparison with other algorithms

	MLP	CNN	LSTM	CNN-LSTM	LSTM-CNN
Learning Rate	0.01	0.01	0.01	0.01	0.01
Number of Neuron	60	50	90	40(CNN) 90(LSTM)	90(LSTM) 40(CNN)
Number of Epoch	180	130	60	110	110

In order to evaluate performance of proposed model, focus was first set on the optimized parameter. For number of neuron, the optimized numbers for all algorithms were different. The neurons for MLP were set in range of 40 to 60. As for LSTM, the neuron numbers were set to 90 as the larger the number of neuron the better the accuracy performance of LSTM. Besides, memory structure in LSTM learns better if the number of neuron increases. CNN on the other hand needed only 50 to create the best performance. In aspect of epoch, LSTM required smaller

number than MLP and CNN, which was 60. In comparison, MLP and CNN needed more number of epoch required by LSTM to achieve the best detection accuracy in which MLP 180 and CNN 130. All these three algorithms acted differently to epoch where LSTM increased almost constantly while MLP and CNN acted the opposite way. The only identical optimized parameter used for all algorithms was the value learning rate parameter. All algorithms prefer lr=0.01. These optimized parameters were then used in stacking model of CNN-LSTM and LSTM-CNN. Both shared the same parameters but differed in layer order in which CNN-LSTM model stacked CNN on top of LSTM layer to handle spatial problem of dataset while LSTM-CNN set LSTM in the first layer to deal with temporal data. The optimized lr, number of neuron and number of epoch used for these combination model were lr=0.01, 90 neuron of LSTM, 40 filter of CNN and 110 epoch respectively.

For better evaluation, this research's model performance was compared with other NN model by setting each algorithm according to their optimized parameter so that their detection performance can be observed and compared fairly. The detection result was tabulated in Table 3 and illustrated in Fig. 8.

Table 3 Comparison accuracy detection of NN model

Neural Network Model	Accuracy (%)±std
MLP	94.73±0.64
CNN	87.91±2.57
LSTM	95.90±0.34
CNN-LSTM	96.76±1.24
LSTM-CNN	**98.53±0.24**

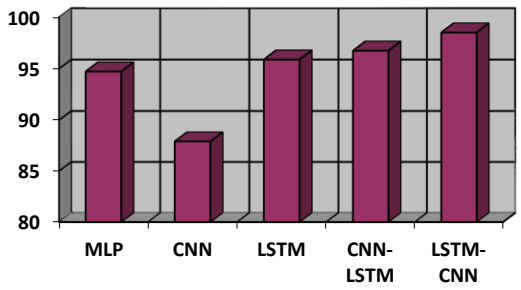

Fig. 8 Accuracy of Detection Model

It was found that among common NN model, LSTM showed the best accuracy while CNN showed the lowest accuracy. The results were displayed as such because of the difference architecture of each algorithm. Basic architecture of LSTM and CNN were MLP but they were strained with special structure. Besides, both CNN and LSTM were invented to exploit temporal invariant in detection. However, in the context of temporal sequences, LSTM outperform MLP and CNN by accurately detecting 95.90% of malware while MLP and CNN detect 94.73% and 87.91% respectively. The main difference of CNN from MLP and LSTM is the layer of

convolution and pooling where it coalescing input data using learned function. This means only selected input features by CNN were chosen to pass as new input to LSTM classifier. LSTM used its memory cell to process and connect all information to produce an input. However, it was not as powerful as the detection by LSTM-CNN model that improved malware detection. The CNN-LSTM classifier depicts higher accuracy outperforming general CNN, LSTM, and MLP. This is because of the existence of best feature selections made by CNN algorithm at the first layer before passing it to LSTM to learn the extracted features. The main difference between CNN-LSTM and LSTM-CNN model was which algorithm came first at the first layer. The detection rate using LSTM-CNN was 98.53% and CNN-LSTM was 96.76% with both detections higher than Drebin detector scheme itself which was 94%. In LSTM-CNN, LSTM layer was the first to receive input, process and stores information not only for current input but also from previous input. The new output produced by LSTM was then fed into CNN to be convoluted where the features of input were extracted.

T-test approach is used to measure the significant difference of the proposed model. The *p*–value or *probability*-value for accuracy performance is calculated, where the hypothesis null is true. Hypothesis null in this work defined as there is no difference between the proposed detection models with existing detection model. If the *p*-value less than 0.5, the hypothesis is rejected and defined as there is significance difference between proposed models with existing models. LSTM-CNN detection model significantly outperforms the standard MLP ($p < 0.0001$), CNN ($p < 0.0001$) and LSTM ($p < 0.0001$). The result supported this study's motivation to combine the neural network algorithm to deal with and detect spatial and temporal problem better. However, the order in stacking the layers played a huge role in model detection performance. By stacking LSTM layer before CNN to process input, better information that accumulated from all inputs were generated before the input were sent to be extracted in CNN for better accuracy. If CNN initiate the layer, the sequence information from the inputs were lost and LSTM function was not fully utilized.

5 Conclusion

This paper reported the study of two malware detection models that combining LSTM with CNN and evaluated the models on the Drebin dataset. Two additional features from the Drebin dataset were extracted and new numerical vector dataset was generated using BOW model that destroyed the spatial information of data. Previous work showed that CNN was well known with its structure in extracting the feature and LSTM was highly capable with temporal modeling. Hence, this study decided to combine LSTM together with CNN and MLP to see if LSTM can help in dealing with spatiotemporal data provided by BOW model representation.

LSTM and CNN were combined as feature classifier and MLP as output classifier. The effect of changing parameter values in the performance of detection was then observed; 10 new sub datasets were generated for testing and performing evaluation. It was found that the LSTM-CNN outperformed MLP, CNN, LSTM and CNN-LSTM in detecting malware. This indicates that hybrid scheme for NN was more accurate and well-suited to detect sequential data. This will be good subject for further research to extend the combination of LSTM and observe its effect on malware detection. In future work, more complex modifications of the LSTM using larger number of samples and more parameters tuning can also be planned and explored. With this modification, it is hoped that the LSTM-CNN model may improve and accurately detect malware better.

ACKNOWLEDGEMENTS

This is a text of acknowledgements The authors would like to thank the Universiti Kebangsaan Malaysia (UKM) and Ministry of Higher Education (MoHe) for their support in making this project possible. This work was supported by the Fundamental Research Grant (FRGS) with grant number FRGS/ 1/ 2016/ ICT02/ UKM/ 02/ 05.

References

[1] J. Saxe & K. Berlin. (2015). Deep neural network based malware detection using two dimensional binary program features. In *10th International Conference on Malicious and Unwanted Software (MALWARE)*, 2015 (pp.11–20).

[2] W. Huang & J. W. Stokes. (2016). MtNet: A multi-task neural network for dynamic malware classification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2016, (vol. 9721, pp. 399–418).

[3] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, & Yiqiang Sheng. (2017). Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, 2017 (pp. 712–717).

[4] Y. Liao & V. R. Vemuri. (2002). Use of k-nearest neighbor classifier for intrusion detection. 2002 *Comput. Secur. (*vol. 21, no. 5, pp. 439–448).

[5] Y. Deng & Y. Zhong. (2013). Keystroke Dynamics User Authentication Based on Gaussian Mixture Model and Deep Belief Nets. 2013 *ISRN Signal Process* (vol. 2013, pp. 1–7).

[6] A. H. Sung & S. Mukkamala. (2003). Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks.

In 2003 *Proceeding. Symp. Appl. Internet*, on (Vol. 1, no. 1, pp. 209–216).

[7] M. Sabhnani, G. Serpen, & K. K. More. (2003). Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context. In 2003 *Proceedings. Int. Conf. Mach. Learn. Model. Technol. Appl. (* pp. 209–215).

[8] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, & J. Nazario. (2007). Automated Classification and Analysis of Internet Malware. In 2007 *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection* on (pp. 178–197).

[9] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, & E. Kirda (2009). Scalable , Behavior-Based Malware Clustering. *Sophia* (vol. 272, no. 3, pp. 51–88).

[10] . S. Pirscoveanu, T. M. T. Hansen, S.S. Larsen, J. M. Stevanovic, M. Pedersen, & A. . Czech. (2015). Analysis of Malware Behavior: Type Classification using Machine Learning. In *Int. Conf. Cyber Situational Awareness, Data Anal. Assess.* on (pp. 1–7).

[11] G. E. Dahl, J. W. Stokes, L. Deng, & D. Yu,. (2013). Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* on (pp. 3422–3426)

[12] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, & A. Thomas,. (2015). Malware classification with recurrent networks. In *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.* on (vol. 2015–Augus, pp. 1916–1920).

[13] B. Kolosnjaji, A. Zarras, G. Webster, & C. Eckert,. (2016). Deep learning for classification of malware system call sequences. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* on (vol. 9992 LNAI, pp. 137–149).

[14] B. Athiwaratkun & J. W. Stokes, . (2017). Malware classification with LSTM and GRU language models and a character-level CNN. In *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process Proc. on* (pp. 2482–2486).

[15] T. N. Sainath, O. Vinyals, A. Senior, & H. Sak. (2015). Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2015 on (vol. 2015 August, pp. 4580–4584)

[16] J. J. Kim, J. J. Kim, H. L. T. Thu, & H. Kim. (2016). Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In

2016 Int. Conf. Platf. Technol. Serv.(pp. 1–5).

[17] P. J. Werbos. (1990). Backpropagation Through Time: What It Does and How to Do It. In *Proc. IEEE* on (vol. 78, no. 10, pp. 1550–1560).

[18] Y. Bengio, P. Simard, & P. Frasconi. (1994). Learning Long Term Dependencies with Gradient Descent is Difficult. In *Trans. Neural Networks* on (vol. 5, no. 2, pp. 157–166). IEEE.

[19] S. Hochreiter & J. Schmidhuber. (1997). Long Short-Term Memory. In *Neural Computer on (*vol. 9, no. 8, pp. 1735–1780).

[20] Y. LeCun & Y. Bengio. (1995). Convolutional networks for images, speech, and time series. In *Handb. brain theory neural networks* (vol. 3361, no. April 2016, pp. 255–258).

[21] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, & K. Rieck. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings 2014 Network and Distributed System Security Symposium*.