Expert Systems With Applications 133 (2019) 151–162

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A multi-level deep learning system for malware detection

Wei Zhong[a,*], Feng Gu[b]

[a] Division of Math and Computer Science, University of South Carolina Upstate, SC 29303, USA
[b] Department of Computer Science, College of Staten Island, 2800 Victory Boulevard, Staten Island, NY 10314, USA

ARTICLE INFO

Article history:
Received 23 September 2018
Revised 27 April 2019
Accepted 28 April 2019
Available online 2 May 2019

Keywords:
Malware detection
Deep learning
Multi-level clustering algorithm
Convolutional neural network
Recurrent neural network
Model construction time

ABSTRACT

To defend against an increasing number of sophisticated malware attacks, deep-learning based Malware Detection Systems (MDSs) have become a vital component of our economic and national security. Traditionally, researchers build the single deep learning model using the entire dataset. However, the single deep learning model may not handle the increasingly complex malware data distributions effectively since different sample subspaces representing a group of similar malware may have unique data distribution. In order to further improve the performance of deep learning based MDSs, we propose a Multi-Level Deep Learning System (MLDLS) that organizes multiple deep learning models using the tree structure. Each model in the tree structure of MLDLS was not built on the whole dataset. Instead, each deep learning model focuses on learning a specific data distribution for a particular group of malware and all deep learning models in the tree work together to make a final decision. Consequently, the learning effectiveness of each deep learning model built for one cluster can be improved. Experimental results show that our proposed system performs better than the traditional approach.

1. Introduction

Malware (short for ***mal***icious soft***ware***) is the general term representing different types of unwanted software programs (e.g., viruses, worms, Trojans, spyware and ransomware) (Hardy, Chen, Hou, Ye, & Li, 2016; Obeis & Bhaya, 2016; Ye, Li, Adjeroh, & Iyengar, 2017). Cyber criminals have used malware to perform malicious activities, such as compromising computers, stealing private information, sending out spam emails, and paralyzing critical infrastructure (Ye et al., 2017). These security attacks often result in severe damages to computer systems and substantial financial losses (Hardy et al., 2016; Ye et al., 2017).

In the fight against malware attacks, one of the traditional malware detection approaches is the signature based method. The signature based methods require manual or automatic generation of signatures that reason over the malware dataset to make detections (Saxe & Berlin, 2015; Ye et al., 2017). Signatures are a short string of bytes, which can uniquely identify each type of known malware so that these types of malware can be accurately detected with a small error rate in the future. The signature based methods can be easily evaded by malware attackers using techniques including encryption, polymorphism and obfuscation (Hardy et al., 2016). Furthermore, malicious files are being generated at a rate of thousands per day, making it challenging for the signature-based methods to be effective and scalable (Hardy et al., 2016; Masud, Khan, & Thuraisingham, 2011a).

In contrast to signature based methods, machine learning techniques can automatically reason about malicious and benign samples to fit the best detection model parameters. Machine learning techniques can help cyber security experts and designers stay one step ahead of cyber attackers instead of just reacting to them (Kazemian & Ahmed, 2015). Recent three trends in the computer community have generated strong interests in applying machine learning techniques to malware detection (Saxe & Berlin, 2015). The first trend is that an increasing number of labeled malware datasets are available to the security community as a result of commercial threat intelligence feeds (Saxe & Berlin, 2015). The second trend is that the computational power becomes cheaper. Consequently, researchers can iterate on machine learning models for malware detection more rapidly and fit larger and more complex deep learning models to the malware dataset (Saxe & Berlin, 2015). The third trend is that researchers have invented more tools to create detection models that have resulted in the breakthrough performance (Ye et al., 2017). The large number of labeled malware samples, cheaper computational powers and the breakthrough in machine learning techniques are the key factors for the security community to re-evaluate the importance of machine learning approaches for malware detection as compared to the traditional signature-based methods.

* Corresponding author.
E-mail addresses: wzhong@uscupstate.edu (W. Zhong), Feng.Gu@csi.cuny.edu (F. Gu).

https://doi.org/10.1016/j.eswa.2019.04.064
0957-4174/© 2019 Elsevier Ltd. All rights reserved.

The shallow learning model and the deep learning model are two main types of machine learning techniques. The shallow learning model usually consists of less than three computational layers. Researchers have applied the shallow learning model, such as Support Vector Machine (SVM), Naïve Bayes (NB), and Decision Tree (DT) for malware detection (Yuan, Lu, & Xue, 2016). These shallow learning models are usually built on an entire dataset. In general, the shallow learning model has not achieved satisfactory performance for malware detection.

Recently, deep learning has attracted increasing attentions in the artificial intelligence community. Deep learning is a type of machine learning that uses many layers of neural network (Chandra & Sharma, 2017; Li, Nie, & Rong, 2018). Deep learning has demonstrated many advantages over shallow learning models in the areas of speech recognition, computer vision, and natural language processing (Papakostasab & Giannakopoulos, 2018; Ronao & Cho, 2016; Wei, hua, Chen, Zhou, & Tang, 2017). Given its wide adoption in other fields, researchers have used deep learning for the development of Malware Detection Systems (MDSs) (Kolosnjaji, Zarras, Webster, & Eckert, 2016; Wei et al., 2017; Yuan et al., 2016). Traditionally, a single deep learning model is built on the entire dataset. This single deep learning model may experience difficulty to handle the increasingly complex data distribution of the malware samples. Some researchers also build a group of deep learning models to enhance the performance of a single deep learning model (Dahl, Stokes, Deng, & Yu, 2013; Shahzad & Lavesson, 2013; Ye et al., 2017). However, this ensemble based approach with multiple deep learning models encounters similar problems as the single deep learning model since each model in the ensemble is still built on the entire dataset.

To further improve the performance of a single deep learning model and the ensemble based multiple deep learning models, we develop a Multi-Level Deep Learning System (MLDLS) that organizes multiple deep learning models using the tree structure. The construction of MLDLS can be divided into five phases. In the first phase, we design a feature selection algorithm to choose thousands of important static and dynamic features (Rhode, Burnap, & Jones, 2017) from billions of candidate features extracted from malware and benignware (short for ***ben***ign soft***ware***). In the second phase, the whole dataset is divided into one-level clusters by the parallel improved K-means clustering. The parallel K-means algorithm takes advantage of the static features and dynamic features selected in the first phase to evaluate the underlying data distribution. In the third phase, the hierarchical clustering algorithm is applied to each one level cluster in parallel to generate the cluster subtrees. Finally, the subtrees are merged to produce the multiple level cluster tree. Basically, the huge dataset is divided into multiple partitions in the multiple levels. Samples in each cluster of the tree usually exhibit similar behaviors and characteristics, which are very useful in identifying highly reused basic patterns of malware variations belonging to the same family (Yue, 2017). In the fourth phase, the best deep learning model is selected for each cluster in the multi-level tree to learn the unique data distribution for each cluster. Several important deep learning models evaluated for each cluster include Convolutional Neural Networks (CNNs), deep Recurrent Neural Networks (RNNs), and deep fully connected Feed Forward Networks (FCs) (Dahl et al., 2013; McLaughlin, Rincon, Kang, Yerma, & Ahn, 2017; Tobiyama, Yamaguchi, Shimada, Ikuse, & Yagi, 2014; Yue, 2017). The best model is selected for each cluster. Using these models, malware patterns can be analyzed from different perspectives. The deep learning model trained for each cluster in the multi-level tree is utilized to learn the unique data distribution for each cluster based on both static and dynamic features. Using multiple machines, training of the deep learning models can be performed in parallel to reduce the processing time. In the final phase, decision values of different deep learning models in the multi-level tree are merged to make the final decision about whether the test sample is malware or not.

In order to evaluate the effectiveness of the malware detection system, the performance of seven computational models is compared: (1) a decision tree built on the entire dataset; (2) a Support Vector Machine built on the entire dataset; (3) a deep fully connected feed-forward neural network built on the entire dataset; (4) a single deep convolutional neural network built on the entire dataset; (5) a single deep recurrent neural network built on the entire dataset; (6) ensemble based deep learning models built on the whole dataset; and (7) a Multi-Level Deep Learning System (MLDLS) built on the tree structure. The ensemble based approach merges the decisions from several single deep learning models constructed on the whole dataset. Three evaluation metrics including True Positive Rate (TPR) or Detection Rate, False Positive Rate (FPR) or False Alarm Rate and the Area Under the Receiver Operating Characteristic Curve (AUC) (Baldi, Brunak, Chauvin, Andersen, & Nielsen, 2000; Ye et al., 2017) are used. Both the combined 5×2 Cross Validation (CV) F test (Alpaydin, 1999) and the independent test are conducted for the rigorous performance evaluation.

The main contributions and the novelty of this work include the following. Firstly, a multi-level deep learning system organizing various deep learning models in the tree structure is designed in order to improve the prediction performance and effectiveness of the traditional single deep learning model built from the entire dataset. Secondly, the proposed multi-level deep learning system is applied to malware detection by dividing the data into multilevel clusters, selecting the best deep learning model for each cluster, and merging decision values of different deep learning models in the tree. Since each deep learning model trained for one cluster is specialized to learn the unique malware data distribution for that cluster, the learning effectiveness of deep learning models can be improved. Thirdly, the computational time of the system is reduced significantly using the parallel computing strategy. Fourthly, parameter tuning process of the multi-level deep learning system for malware detection is discussed in details in order to provide the guidance about how to use the proposed system.

Experimental results in Section 5 demonstrate that different types of deep learning models are suitable for various malware sample subspaces. The traditional one-model-fit-all strategy does not work well for complex malware data distribution. Furthermore, detailed experiments are conducted to compare the performance of the traditional single deep learning model and the proposed multi-level deep learning system in Section 6. The thorough statistical analysis demonstrates that both improvements in prediction accuracy and effectiveness for our proposed system is statistically significant as compared to the traditional single deep learning model built from the entire dataset. The experimental results also show that the ensemble based approach using multiple deep learning models built on the entire dataset also performs worse than our proposed multi-level deep learning system. Due to the parallel strategies, the processing time of our proposed system is comparable to the single deep learning model while achieving improved prediction performance and effectiveness. The experimental results confirm that our proposed approach opens the new avenue to further improve the malware detection system as compared to the traditional approaches.

The rest of the paper is organized as follows. Section 2 discusses the related works. Section 3 describes five phases of MLDSL. Section 4 explains the training set, the testing set, evaluation metrics, and evaluation methods. Section 5 provides details of parameter tuning. Section 6 presents the experimental results and analysis. Finally, Section 7 concludes the paper and points out the future work.

2. Related work

Due to its low false positive rate, traditional signature-based approaches are very popular in anti-malware industry (Ye et al., 2017). However, the true positive rate of signature-based approaches can be reduced noticeably if the malware attackers adopt the evading techniques such as encryption, polymorphism and obfuscation (Sung, Xu, Chavez, & Mukkamala, 2014). In order to effectively and automatically detect malware samples from the large real-world dataset, researchers have used shallow learning models that have less than three layers of computational units such as Support Vector Machine and decision tree (Yuan et al., 2016). These shallow learning models generally cannot take advantage of the large number of malware samples generated every day. Consequently, the performance of the shallow learning model is not satisfactory.

Since the performance of deep learning models keeps improving with the increasingly number of samples (Ye et al., 2017), researchers build a single deep learning model using an entire data (i.e., a dataset that blends data from multiple types of malware into one dataset) in order to learn the relationship between data features extracted from malware and the target (Benchea & Gavriluţ, 2014; Jung, Kim, & Choi, 2015; Pascanu, Stokes, Sanossian, & Marinescu, 2015; Saxe & Berlin, 2015). These deep learning models mainly use three types of neural network architectures: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Fully connected Feedforward Neural Network (FC). The malware executables are represented as the binary vector of zeros and ones. After reshaping the vector into the matrix, the malware samples can be viewed as the gray-scale image (Gilbert, 2016). CNN is specialized in discovering the spatial relationship of byte patterns in malware images. After running the malware executables in the virtual machines, the sequence of API calls is recorded. RNN is particularly designed to analyze the sequence of API calls for malware samples to find out the sequential dependence (Pascanu et al., 2015). The deep Fully Connected Feedforward Neural Network (FC) just uses the regular neural network architecture with many layers.

There are two major disadvantages to building a single deep learning model that uses a blended dataset: 1) complex data distribution and 2) scalability. Each type of malware has unique and different characteristics, proliferation methods and data distributions (Obeis & Bhaya, 2016; Ye et al., 2017). Consequently, merging different types of malware into one dataset results in a very complex overall data distribution. Furthermore, the diversity and sophistication of the merged dataset continue to grow rapidly due to the large number of new malware variants that are created each year (Ye et al., 2017). As a result, it is very challenging for a single deep learning model to understand this complex data distribution. Additionally, the single CNN model treats malware as the image while the single RNN model considers the behavior of malware as the sequence of events. Both models only analyze the data distribution from only one perspective. For more complex malware data distributions, analysis of its data distribution in different sample subspaces from multiple angles is preferred in order to combine knowledge and strength of these single models effectively. Second, building a single deep learning model for malware detection lacks scalability to train on increasingly large malware datasets. Training deep learning models on very large datasets is a computationally expensive process (Gilbert, 2016). Since the number of new malware samples has exponentially increased through time (Benchea & Gavriluţ, 2014; Kolosnjaji et al., 2016), building a single deep learning model requires longer and longer computational time. This slow training process hinders the efforts to search and rebuild the learning model rapidly in order to adapt to the fast changing malware landscape and respond to new techniques generated by malware attackers.

An alternative to use a single deep learning model to build MDSs is the development of ensemble based deep learning models. Multiple deep learning models in the ensemble can work together to enhance the performance of MDSs. Researchers have developed the ensemble based deep learning models (Dahl et al., 2013; Shahzad & Lavesson, 2013; Ye et al., 2017). Each deep learning model in the ensemble based approach is constructed on the whole blended dataset. Consequently, the ensemble based approach faces similar problems of the single deep learning model.

To overcome the potential weakness of the previous approaches, we propose the Multi-Level Deep Learning System (MLDLS) for malware detection. In the proposed MLDSL, each deep learning model in the tree structure is not built on the whole blended dataset. Instead, it is built on the subset of data for a particular group of malware family. Thus, each model can focus on learning a specific data distribution for a particular group of malware and all deep learning models in the tree work together to make a final decision. As compared to previous studies using deep learning for malware detection, MLDLS is more capable to handle increasingly complex malware data distributions. Our MLDLS also addresses MDS scalability issues by utilizing the multi-level clustering algorithm and parallel training for multiple deep learning models. We will provide the details of the multiple-level deep learning model in the following sections.

3. Multi-level deep learning system for malware detection

The construction of MLDLS is divided into five phases: Phase 1: Selecting important static and dynamic features from the candidate feature set; Phase 2: Partitioning the dataset into multiple one-level clusters using the parallel improved K-means algorithm; Phase 3: Generating multiple cluster subtrees in parallel; Phase 4: Building the deep learning model for each cluster in the tree in parallel; Phase 5: Combining decision values of deep learning models in the tree to classify samples as malware or benign. The flowchart to construct the MLDLS model is shown in Fig. 1.

3.1. Static and dynamic feature selection

During the n-gram analysis, each n-gram is generated by sliding a window of the size n and the best n-grams are selected based on information gains. The concept of n-gram is expanded from byte sequences to kernel API call (system call) sequences. In other words, an n-gram may be either a subsequence of n bytes extracted from binary executables or a subsequence of n API calls extracted from the log of execution sequences after running malware in the virtual machines. The subsequences of n bytes obtained without executing binary executables are called static features (Gilbert, 2016). The subsequences of n API calls obtained after analyzing the behavior of binary executables are called dynamic features (Gilbert, 2016).

It is generally inappropriate to use all n-gram static and dynamic features extracted from binary executables. When samples are divided into multiple one-level clusters using the exponential number of such binary n-gram features, the effectiveness of the clustering algorithm can be reduced considerably since most of these features may be noisy, redundant or irrelevant. In order to avoid these problems, candidate n-gram features need to be sorted based on certain criteria so that a small subset of features with the greatest discriminatory power can be selected for the model construction.

The information gain (Han, Kamber, & Pei, 2011) is used as the selection criterion in this work since it is one of the most effective feature selection measures reported in the literature (Masud et al., 2011b). In this work, the information gain is utilized to evaluate the effectiveness of a feature to classify the training data. If the

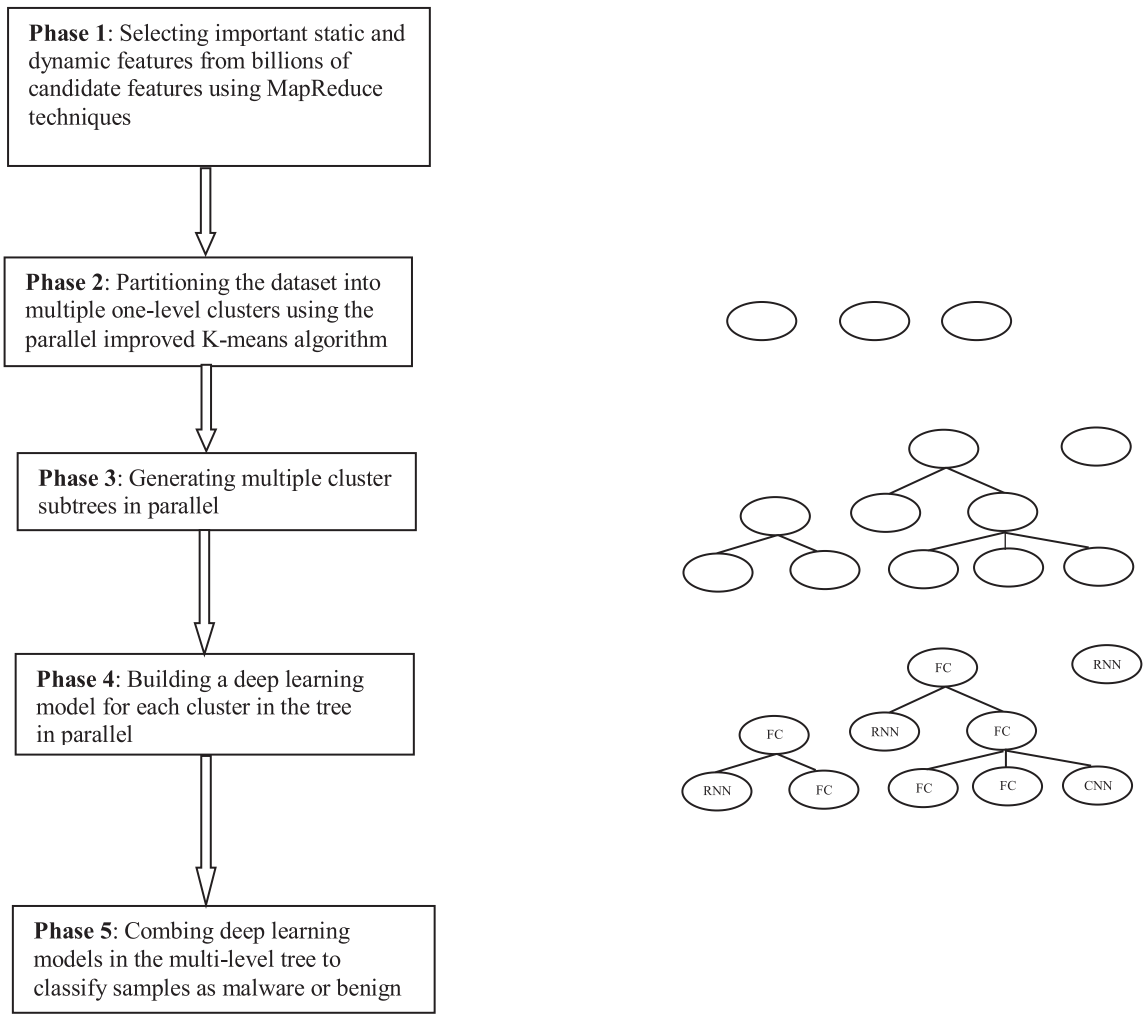
Phase 1: Selecting important static and dynamic features from billions of candidate features using MapReduce techniques
Phase 2: Partitioning the dataset into multiple one-level clusters using the parallel improved K-means algorithm
Phase 3: Generating multiple cluster subtrees in parallel
Phase 4: Building a deep learning model for each cluster in the tree in parallel
Phase 5: Combing deep learning models in the multi-level tree to classify samples as malware or benign
FC
RNN
FC
RNN
FC
RNN
FC
FC
FC
CNN

Fig. 1. The flow chart for building the MLDLS Model.

training data is split based on values of this feature, the information gain measures the expected reduction in entropy after the split. The feature with the higher information gain is more effective to classify the sample. The information gain can be defined as Eq. (1) (Han et al., 2011):

$$Gain(A) = Info(D) - Info_A(D) \tag{1}$$

where *Info*(D) is the original information requirement and *Info*$_A$(D) is the new requirement after splitting on the feature A. *Info*(D) is defined as Eq. (2):

$$Info(D) = -\frac{pos}{total}\log_2\frac{pos}{total} - \frac{neg}{total}\log_2\frac{neg}{total} \tag{2}$$

where *pos* is the total number of positive (malware) samples in the dataset, *total* is the total number of samples in the dataset, and *neg* is the total number of negative (benign) samples in the dataset. *Info*$_A$(D) is defined as Eq. (3):

$$Info_A(D) = \sum_{v\in(0,1)} \frac{total_v}{total}\left(-\frac{pos_v}{total_v}\log_2\frac{pos_v}{total_v} - \frac{neg_v}{total_v}\log_2\frac{neg_v}{total_v}\right) \tag{3}$$

The information gain ratio is defined as Eq. (4)

$$InfoGainRatio(A) = \frac{Gain(A)}{Info(D)} \tag{4}$$

The feature extraction and selection process can be time consuming and space intensive for very large datasets. In our work, billions of candidate static features are generated. In order to speed up the feature selection process, the Map-Reduce algorithm is developed. The training dataset is first distributed among machines in the Hadoop Distributed File System (HDFS). Feature extraction and information gain calculation are conducted independently in each machine. Finally results from all machines are combined together. This algorithm is implemented based on the previous work (Masud et al., 2011b). After important static and dynamic features are extracted from binary executables, one-level clusters and multi-level clusters are built based on these features.

3.2. Dataset partition

The traditional K-means (Gupta, Rao, & Bhatnagar, 1999) method has two potential weaknesses for huge datasets: (1) random selection of initial cluster centers may lead to distorted or improper partitions, which are far away from the globally optimal solution; (2) scalability is a major concern. For example, a large percentage of samples may be grouped into small numbers of clusters while remaining clusters have a few samples as a result of the random selection of initial cluster centers. How to process huge amount of samples efficiently is another big issue for the K-means clustering algorithm (Owen, Anil, Dunning, & Friedman, 2011).

In order to solve the potential problems related to random selection and scalability, the parallel improved K-means clustering using Apache Spark is proposed in this work. The parallel improved

K-means clustering algorithm uses both static and dynamic features extracted in the previous step. Each iteration of the parallel K-means clustering algorithm can be divided into two phases, the mapping phase and the reducing phase. During the mapping phase, the closest centroid for each sample in different partitions is calculated in parallel. During the reducing phase, the centroid for each cluster is recalculated after obtaining the partial sum of the centroid from each partition for one cluster. The parallel improved K-means algorithm can potentially process millions of samples efficiently.

To solve the potential weakness of random initialization, the greedy initialization technique is included into the parallel K-means clustering algorithm. The greedy initializing techniques are intended to select appropriate initial points so that final clustering results can represent the underlying data distribution more precisely and consistently. In the greedy initializing method, the parallel K-means clustering algorithm will only be carried out for several iterations during each run. After each run, initial points, which are used to generate the cluster with high quality, are selected. The quality of clusters is defined as Eq. (5):

$$Cluster_{Quality}(\%) = \max\left(P_{Benign}, P_{Malware}\right) \tag{5}$$

where P_{Benign} is the percentage of benign samples in the given cluster and $P_{malware}$ is the percentage of malware samples in the given cluster. For example, if the percentage of benign samples in the given cluster is 20% and the percentage of malware samples in the same cluster is 80%, the cluster quality of this cluster is 80%.

If the minimum distance of new points to all points already selected in the initialization array is greater than the given threshold, these points will be included into the initialization array. The distance score between two samples is defined as Eq. (6):

$$Dist(x, y) = \sum_{i=1}^{N} |F_x(i) - F_y(i)| \tag{6}$$

where N is the number of features for each sample. $F_x(i)$ is the value of the feature at index i for the sample x. $F_y(i)$ is the value of the feature at index i for the sample y.

The minimum distance examination makes sure that each recently chosen point can potentially belong to different natural clusters. This procedure will be repeated until the specified number of initial points are selected. After this process, the selected points are used as the initial cluster centers for the parallel improved K-means algorithm.

3.3. Multiple level cluster tree generation

Since one-level clusters may not capture all valuable underlying data distributions, the parallel multi-level cluster tree algorithm is used to discover high quality subclusters from one-level clusters. In the first step of the parallel multi-level cluster tree algorithm, the agglomerative hierarchical algorithm is carried out for each one-level cluster. The cluster merging process repeats until the quality of the merged clusters falls below the given threshold. After the first step, a forest of cluster subtrees is produced. In the second step, the agglomerative hierarchical clustering algorithm is carried out to merge the root clusters of each subtree. Again the cluster merging process repeats until the quality of the merged clusters falls below the given threshold. Taking advantage of the multi-level tree structure, clusters at different levels can capture different underlying data distribution patterns for a particular subspace of data.

3.4. Deep learning model training

The multi-level tree structure is able to find out some high-quality subclusters from one-level clusters. However, the noisy and irrelevant information is still introduced into clusters in the multi-level tree structure, which may significantly reduce the performance of the malware detection system. In order to enhance the detection performance of the multi-level tree, each deep learning model is trained to recognize subtle patterns distinguishing between malware and benign samples for one cluster in the multi-level cluster tree based on both dynamic and static features. During the training process, several important deep learning models evaluated for each cluster include Convolutional Neural Networks (CNNs), deep Recurrent Neural Networks (RNNs), and deep Fully Connected Feed Forward networks (FCs). Only one best model is selected for each cluster.

These three types of models can be used to analyze malware patterns from different perspectives. Recurrent Neural Network (RNN) can explicitly model the sequential dependencies in the API call sequences because the output relies on previous inputs (Tobiyama et al., 2014). To avoid the vanishing gradient problem for very long sequences, the Long Short-Term Memory (LSTM) is incorporated (Hochreiter & Schmidhuber, 1997) into the RNN model. The LSTM unit can not only reduce the gradient vanishing problem by fixing weights of hidden layers but also retain relevant input information, which is required for future outputs (Tobiyama et al., 2014). Fig. 2 shows the unrolled diagram with time of the 4-layer Recurrent Neural Network (RNN) with LSTM. The RNN in this work has four computational layers. Each layer uses the LSTM unit to capture the long-term dependence.

The Convolutional Neural Network (CNN) consists of multiple convolution and pooling layers, several dense layers and the softmax output layer (Gilbert, 2016). As you go deeper into the CNN, the height and width of the image file usually decrease and the number of channels usually increase (Gilbert, 2016). Fig. 3 shows the diagram of the convolutional neural network. The convolution layers can learn new pattern detectors automatically from raw data after discovering the correlation between neighboring input feature vectors (Kolosnjaji et al., 2016). This approach effectively removes requirements of enumerating a large number of n-gram features since CNN can intrinsically learn n-gram pattern detectors for malware during the training (Kolosnjaji et al., 2016). Since CNN can be used to discover the higher-order local features that are invariant to small changes in data, it is robust against instruction reordering and incorporation of unreachable codes (Kolosnjaji et al., 2016).

In contrast to RNN and CNN models, the feed forward neural networks do not assume the structure of samples and do not share features learned across different positions of one sample.

These three types of deep neural networks all implement a mapping function from the input vector x to the output vector y using a sequence of layers. The feedforward function for each layer of neural networks is defined as Eq. (7) (Dahl et al., 2013):

$$a_{i+1} = f_i(W_i \times a_i + b_i) \tag{7}$$

where a_i is the activation of the ith layer, W_i is the weight matrix for ith layer, and b_i is the bias for ith layer.

The loss function for all 3 types of neural networks is defined as Eq. (8) (Gilbert, 2016):

$$Loss\left(y, \hat{y}\right) = -\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1 - y_i)\log \hat{y}_i\right] \tag{8}$$

where y_i is the true label for sample i. $y_i \in \{0, 1\}$ with 0 representing benign sample and 1 malware. $\hat{y}_i$ is the output of our deep learning model for sample i. N is the size of one batch.

The deep neural networks are trained to minimize this loss function using backpropagation algorithm (Krizhevsky, Sutskever, & Hinton, 2012). Thus, the formula to update the parameters us-

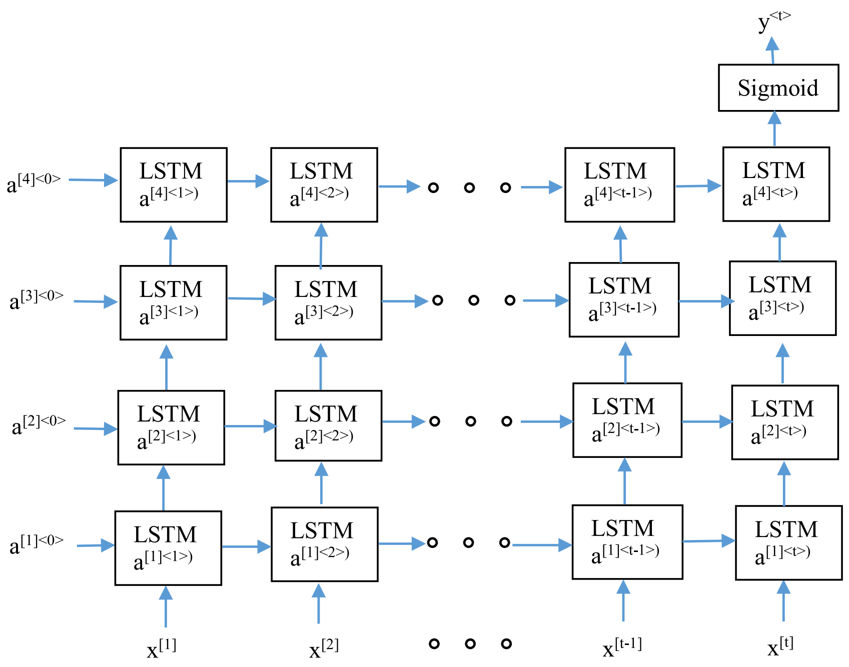

Fig. 2. Unrolled diagram for Recurrent Neural Network (RNN) with LSTM.

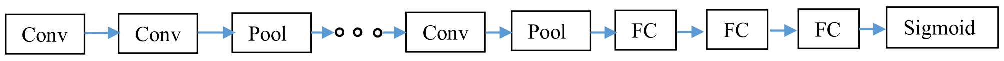

Fig. 3. Diagram for convolutional neural network.

ing one iteration of gradient descent is defined as Eq. (9):

$$w = w - \alpha \frac{\partial L(y, \hat{y})}{\partial w} \tag{9}$$

where α is the learning rate.

Each deep learning model can focus on highly related samples in each cluster without being confused by noisy or irrelevant information from other clusters. Consequently, the learning efficiency for each deep learning model can be improved.

3.5. Advanced decision fusion algorithm

The deep learning decision function for the cluster k to classify sample x is formulated as Eq. (10):

$$f_{dl\_k}(x) = sigmoid(W_k \times y_k + b_k) \tag{10}$$

where y_k is the “activation” of the last layer of the neural network for the cluster k and W_k is the weight matrix between the last layer and the output layer of neural networks for cluster k. In this work, the decision score reflects how confident the deep learning model assigns the sample as benign or malware based on static and dynamic features.

In order to compare classification values from different deep learning models fairly and objectively, the classification value, $f_{dl\_k}(x)$, of a deep learning model, dl_k, is normalized using the z-score. This normalization procedure is essential since the decision function of deep learning models for different clusters in the tree structure is calculated from various high-dimensional feature space.

The *decision value* of dl_k for sample x is formulated as the z-score of dl_k's classification value for a given sample x in Eq. (11):

$$decision\_value_{dl\_k}(x) = \frac{(f_{dl\_k}(x) - mean_{dl\_k})}{\sigma_{dl\_k}} \times training\_accuracy_{dl\_k} \tag{11}$$

where $mean_{dl\_k}$ is the mean classification values for dl_k in cluster k and σ_{dl_k} is the standard deviation of classification values for dl_k in the cluster k. Higher magnitude of the *decision value* of a dl_k, $|Decision\_value_{dl\_k}(x)|$ indicates that the deep learning model is more confident to classify a given sample. Since the distance between the sample and the cluster related to this deep learning model can influence the confidence level of the deep learning decision value, the deep learning decision value for the cluster is weighted by the distance value between the sample and the cluster linked to the deep learning model as defined in Eq. (12).

$$dist(k, x) = \sum_{i=1}^{N} |F_x(i) - F_k(i)| \tag{12}$$

where N is the number of dynamic and static features for each sample. $F_x(i)$ is the value of dynamic and static feature at index i for the sample x. $F_k(i)$ is the value of dynamic and static feature at index i for the centroid of cluster k. The logistic function used to smooth the distance between the sample x and cluster k is defined as Eq. (13):

$$smooth\_dist(k, x) = \frac{1}{1 + e^{-dist(k,x)}} \tag{13}$$

where k is the cluster k and x is the given sample. As a result, the *weighted decision value* of dl_k for a sample x is defined as Eq. (14):

$$\psi(dl\_k, x) = decision\_value_{dl\_k}(x) \times smooth\_dist(k, x) \tag{14}$$

If the training accuracy of deep learning models for the given cluster is below the given threshold, they will not be included into the decision making process. Finally the highest weighted decision value is used to decide whether the sample is malware or not. Fig. 4 shows the detailed algorithm to build the MLDLS model.

Multi-Level Deep Learning System (MLDLS)

Phase 1. Selecting thousands of important dynamic and static features using information gains and MapReduce techniques.

Phase 2. Partitioning the whole dataset into multiple one-level clusters using the parallel improved K-means clustering algorithm. The clustering algorithm uses both dynamic and static features.

```
WHILE (the number of initial points discovered is less than the required number of clusters)
{
    K initial points are randomly selected.
    While(the running iteration is less than a fixed number of iteration){
        The closest centroid for each sample is calculated in parallel using multiple machines in Apache Spark
        Infrastructure.
        Cluster center for each cluster is reevaluated.
    }
    END WHILE
    Assess the quality of clusters created by each initial point
    IF (the quality for one cluster is better than a given threshold)
    {
        Check the minimum distance of the point creating this cluster with existing points in
        the initialization array
        IF (the minimum distance is bigger than threshold)
            This new point is included into the initialization array
        END IF
    }
    END IF
}
END WHILE
Run the parallel K-means algorithm using the carefully selected Initial Points in the Apache Spark
infrastructure
```

Phase 3. Generating multiple cluster subtrees

```
FOR each one-level cluster
{
    Applying the agglomerative hierarchical clustering algorithm in parallel. Merging of two clusters stops
    when the quality of the merged cluster falls below the given threshold. In the end, this step produces a
    tree structure for one cluster.
}

Applying the agglomerative hierarchical clustering algorithm to the root clusters of each tree structure.
Merging of two clusters stops when the quality of the merged cluster falls below the given
threshold.
```

Phase 4. Training and selecting the best deep learning model for each cluster in the cluster tree in parallel

Phase 5. Selecting the most suitable deep learning model from the cluster tree and classifying samples as malware or benign

Adjusting the decision value of the deep learning model by the distance score

$$\psi(dl\_k, x) = decision\_value_{dl\_k}(x) \times smooth\_dist(k, x)$$

Fig. 4. Five phases of the MLDLS model.

4. Datasets and experimental setup

In this section, datasets for the combined 5×2 Cross Validation (CV) F test and independent test are discussed first. Then the details of performance metrics are explained.

4.1. Dataset for combined 5×2 cross validation F test and independent test

Our malware dataset consists of samples gathered from six primary sources: Virus Share, Maltreive, VX Heavens, Offensive Computing, VirusSign (Maxwell, 2016; Offensive Computing, 2017; Roberts, 2016; Saxe, 2017; VirusSign, 2017; VX Heaven, 2016) and a private collection. These sources provide a large and diverse amount of malware samples. The benignware samples are obtained from various reputable sources to generate representative distributions of benign samples (Obeis, & Bhaya, 2016; Ye et al., 2017). All malware and benignware samples are Window executables, which are one type of the primary targets for cyber criminals. In total, we developed a dataset with 2242,234 (2.2 million) malware samples and 3425,176 (3.4 million) benignware samples. Eighty percent of the samples in our dataset are randomly selected for use in the Combined 5×2 Cross Validation F Test (Baldi et al., 2000). The remaining 20% of samples in the dataset serve as the independent testing set.

4.2. Performance evaluation metrics

True Positive Rate, False Positive Rate, and the Area Under the Receiver Operating Characteristic Curve (AUC) (Baldi et al., 2000; Ye et al., 2017) are used to evaluate the performance of malware detection system. True Positive Rate (TPR) is defined as

Table 1
Information gain ratio thresholds vs. average number of static features, average TPR and average AUC during the combined 5 × 2 CV F test.

Information gain ratio threshold	# static features	TPR	AUC
0.01	112,348	72%	68%
0.03	24,307	81%	76%
0.05	3211	92%	85%
0.07	1272	89%	85%
0.09	87	83%	81%

Table 2
Information gain ratio vs. average number of dynamic features, average TPR and average AUC during the combined 5 × 2 CV F test.

Information gain ratio threshold	# dynamic features	TPR	AUC
0.01	2527	87%	89%
0.03	421	89%	90%
0.05	195	84%	86%
0.07	79	72%	75%
0.09	52	68%	69%

Eq. (15) (Obeis, & Bhaya, 2016; Ye et al., 2017):

$$TPR = \frac{TP}{TP + FN} \tag{15}$$

where *True Positive* (TP) represents the number of malware samples that are correctly recognized and *False Negative* (FN) represents the number of malware samples that are incorrectly recognized as the benign samples. TPR is also called the detection rate. False Positive Rate (FPR) is defined as Eq. (16) (Obeis & Bhaya, 2016; Ye et al., 2017):

$$FPR = \frac{FP}{FP + TN} \tag{16}$$

where *False Positive* (FP) represents the number of benign samples that are incorrectly recognized as the malware samples and *True Negative* (TN) represents the number of benign samples that are correctly recognized. *FPR* is also called the false alarm rate. The Area Under the Receiver Operating Characteristic Curve (AUC) (Baldi et al., 2000) is based on the set of TPR and FPR pairs produced from different thresholds. Since our dataset is slightly imbalanced, TPR and AUC are the traditional approach to evaluate the imbalanced dataset for malware detection (Saxe, & Berlin, 2015; Turki, & Bhaya 2016; Ye et al., 2017). TPR is also called the detection rate.

5. Parameter tuning for deep learning system

In this section, the parameter tuning for MLDLS is discussed in details.

5.1. Static and dynamic feature selection

In Table 1, the average number of static features, the average TPR and the average AUC for different information gain ratio thresholds during the combined 5 × 2 CV F test is demonstrated. In Table 2, the average number of dynamic features, the average TPR and the average AUC for different information gain ratio thresholds during the combined 5 × 2 CV F test is revealed. The best set of dynamic and static features are combined during the clustering process.

5.2. Optimal one-level cluster number selection

In Fig. 5, the average performance for different number of one-level clusters during the combined 5 × 2 CV F test is compared. When the number of one-level cluster is 50, the classification performance almost reaches the peak.

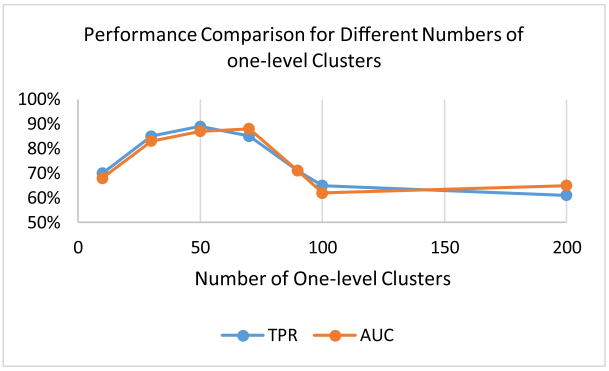

Fig. 5. Average performance for different number of one-level clusters during the combined 5 × 2 CV F test.

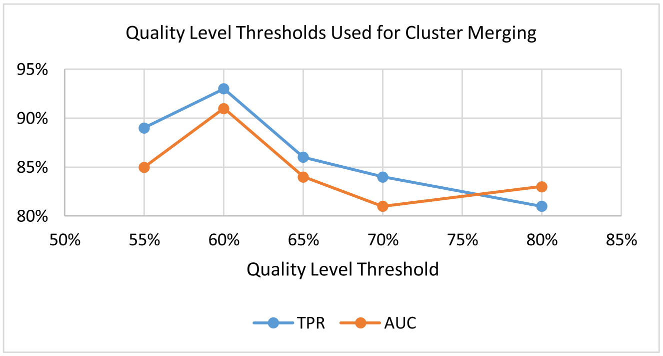

Fig. 6. Quality Level Threshold used for cluster merging during multiple level cluster tree generation.

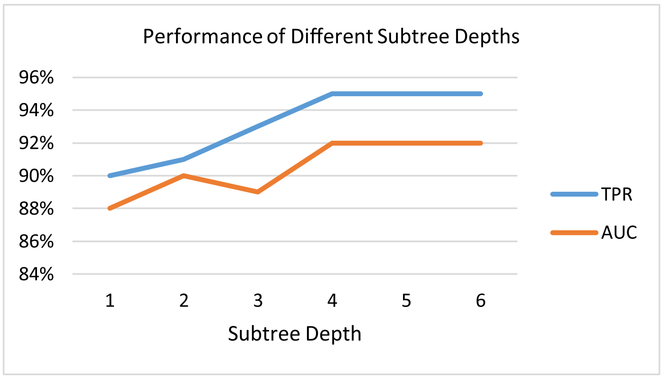

Fig. 7. Performance of different subtree depths during the combined 5 × 2 CV F test.

5.3. Parameter selection for multi-level clustering

In Fig. 6, the quality level threshold used to guide the cluster merging process for the multiple level cluster tree generation is analyzed. Fig. 6 demonstrates that 60% of the quality level threshold produces the best prediction result. In Fig. 7, the performance of the system using different levels of subtrees is compared. After the depth of subtree reaches four, the performance gains stop.

5.4. Model selection for different clusters in the subtree

In Table 3, model configurations of Convolutional Neural Networks are demonstrated. In Table 4, model configurations for fully Connected Neural Network (FCs) are described. In Table 5, model configurations for Recurrent Neural Network (RNN) are shown. In Fig. 8, the average percentage of different deep learning models for each subtree is shown. The activation functions for all models are

Table 3
Model configuration of convolutional neural network.

Model configuration 1	Model configuration 2	Model configuration 3
Input layer of 128×128 based on the malware image from original binary executable file		
1 Conv layer (64 3 × 3 filters)	2 Conv layers (64 3 × 3 filters)	3 Conv layers (64 3 × 3 filters)
Max-pooling layer	Max-pooling layer	Max-pooling layer
1 Conv layer (128 3 × 3 filters)	2 Conv layers (128 3 × 3 filters)	3 Conv layers (128 3 × 3 filters)
Max-pooling layer	Max-pooling layer	Max-pooling layer
1 Conv layer (256 3 × 3 filters)	2 Conv layers (256 3 × 3 filters)	3 Conv layers (256 3 × 3 filters)
Max-pooling layer	Max-pooling layer	Max-pooling layer
1 FC layer (512 neurons)	2 FC layers (512 neurons)	3 FC layers (512 neurons)
Sigmoid output layer	Sigmoid output layer	Sigmoid output layer

Table 4
Model configurations of fully connected neural network.

Num of Layers	Num of neurons
4	[128,128,128,128]
8	[256,256,256,256,128,128,128,128]
10	[512,512, 256,256,256,256,128,128,128,128]

Table 5
Model configuration of recurrent neural network.

Hyper parameters	LSTM-1	LSTM-2	LSTM-3
Depth	1	2	3
No. of neurons	64	128	256

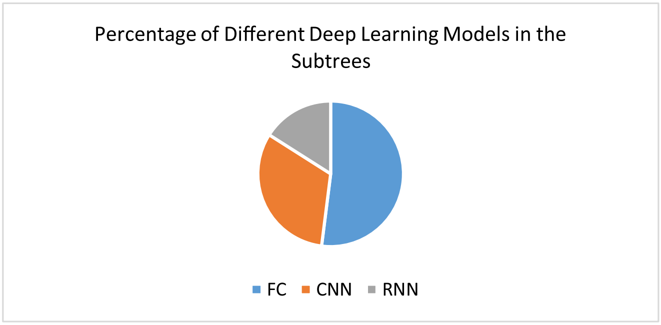

Fig. 8. Average percentage of different deep learning models in the subtree during the combined 5 × 2 CV F test.

Leaky Relu. The dropout rate is 0.1 to prevent overfitting. The combined 5 × 2 CV F test is used to select the best model configuration for each cluster.

5.5. Summary for parameter tuning

Fig. 7 indicates that the performance of malware detection system can be improved while the depth of subtree increases. Fig. 8 clearly demonstrates that the most suitable deep learning model for different sample subspaces is quite different. For different sample subspaces, various types of deep learning can be specialized to learn the unique data distribution. The traditional approach of building the single deep learning model from the entire dataset is not suitable to capture all unique data distributions in different subsamples. Our results also show that the traditional approach of fitting one deep learning model for quite diverse data distributions cannot handle the increasingly complex malware data distributions.

6. Experimental results and analysis

In this section, experimental results for the combined 5 × 2 CV F test and the independent test are used to compare the performance of several computational models. The construction time for our computational model is also reported.

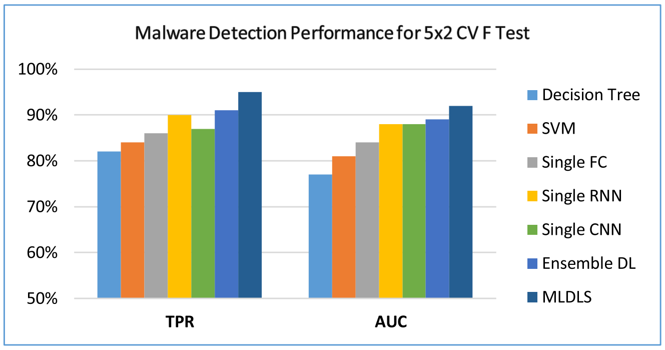

Fig. 9. Malware detection performance for combined 5 × 2 CV F test.

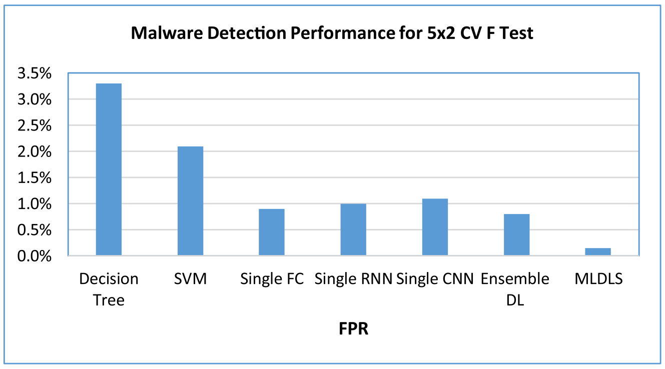

Fig. 10. Malware detection performance for combined 5 × 2 CV F test.

6.1. Malware detection performance for the 5 × 2 CV F test

At first, experimental results for comparing the malware detection performance of seven models are discussed. In Fig. 9, the malware detection performance for seven models using the 5 × 2 CV F test is compared. In Fig. 10, the False Positive Rate (False Alarm Rate) for seven models using the 5 × 2 CV F test is compared. The label “Decision Tree” denotes the single decision tree model constructed from the entire dataset. The label “SVM” denotes the single Support Vector Machine model constructed from the entire dataset. Both the decision tree and SVM are the shallow learning model. The label “Single FC” denotes the single deep fully connected feedforward neural network built on the entire dataset. The label “Single RNN” denotes the single deep recurrent neural network built on the entire dataset. The label “Single CNN” denotes the single deep convolutional neural network built on the entire dataset. The label “Ensemble DL” denotes the ensemble-based deep neural networks built on the entire dataset. The label “MLDLS” denotes the Multi-Level Deep Learning System proposed

Table 6
"P-value by F test" for seven models.

Model	FPR	TPR	AUC
Decision Tree	<0.1%	<0.1	<0.1%
SVM	<0.1%	<0.1	<0.1%
Single FC	<0.1%	<0.1	<0.1%
Single RNN	<0.1%	0.8%	<0.1%
Single CNN	<0.1%	0.3%	0.6%
Ensemble DL	0.5%	0.7%	0.4%
MLDLS	N/A	N/A	N/A

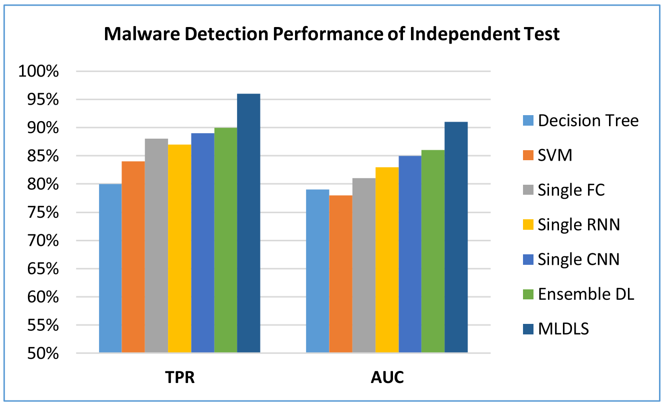

Fig. 11. Malware detection performance of seven models for independent test.

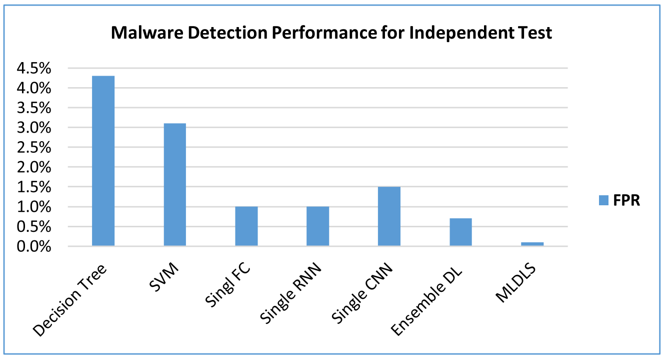

Fig. 12. Malware detection performance of seven models for independent test.

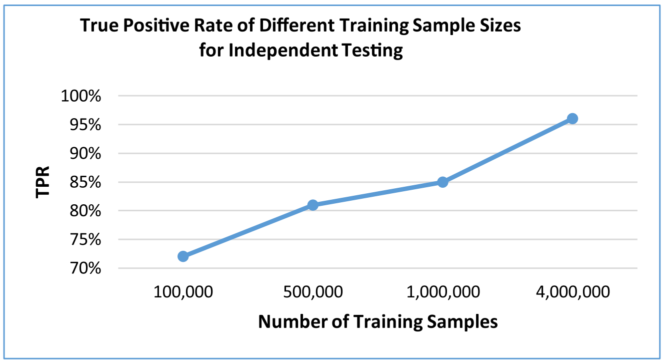

Fig. 13. True positive rate of MLDLS constructed from different training sample sizes.

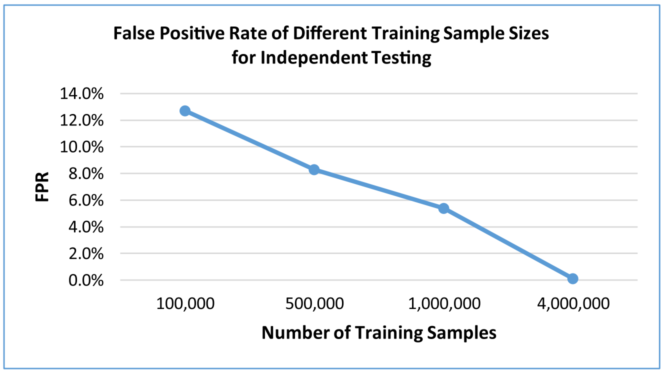

Fig. 14. False positive rate of MLDLS constructed from different training sample sizes.

in this work. Our results show that in general the performance of deep learning model is better than that of the shallow learning model such as decision tree and SVM. Compared with the Ensemble DL (ensemble-based deep learning models), MLDLS improves *TPR* and *AUC* by 4 and 3 percentage points respectively. MLDLS also outperforms all the single deep learning models built from the blended whole dataset. This demonstrates that the multi-level deep learning system can further improve the malware detection performance. Compared with other 6 models, False Positive Rate for MLDLS has been reduced noticeably.

The combined 5×2 Cross Validation (CV) F test is carried out to find out whether the malware detection performance improvement of MLDLS over other six models is statistically significant. The p-value generated from the combined 5×2 CV F test indicates the significant level at which the null hypothesis that algorithm has the same error rate can be rejected. A lower p-value generally suggests a more statistically significant improvement of MLDLS over the other six models. In this work, the significant level for p-value is set to 1%, which is more rigorous than 5% normally selected by statisticians. Table 6 indicates "p value by F test" when seven computational models are compared. Experimental results from Table 6 show that performance improvement of MLDLS over the other six computational models in terms of all performance metrics is statistically significant.

6.2. Experimental results for independent test

Fig. 11 compares the malware detection performance of seven models on the independent testing set. Compared with the ensemble DL, the MLDLS improves *FPR* and *AUC* by 6 and 5 percentage points respectively. The MLDLS also performs much better than the single deep learning model for the independent testing. Fig. 12 compares the False Positive Rate (False Alarm Rate) for seven models on the independent testing set. The similar performance improvement for MLDLS is observed as compared with other models.

In order to investigate whether including more training samples for the model construction can lead to more actionable information for malware detection systems, the performance of MLDLS constructed from different training sample sizes is compared. Fig. 13 compares the True Positive Rate of MLDLS constructed from different sample sizes, using the independent testing set. Fig. 14 compares the False Positive Rate of MLDLS constructed from different sample sizes, using the independent testing set. Both figures show that increasing the number of samples for building the model actually improves the performance of malware detection model significantly.

6.3. Model construction time for different number of machines

Since training of deep learning models is computationally expensive, deep learning models for different clusters in the multi-level cluster tree are trained in parallel. Fig. 15 indicates the average model construction time (in hours) of MLDLS for 5×2 CV F test when different numbers of machines are used. The model construction time for MLDLS is 95 h when the 128 machines are

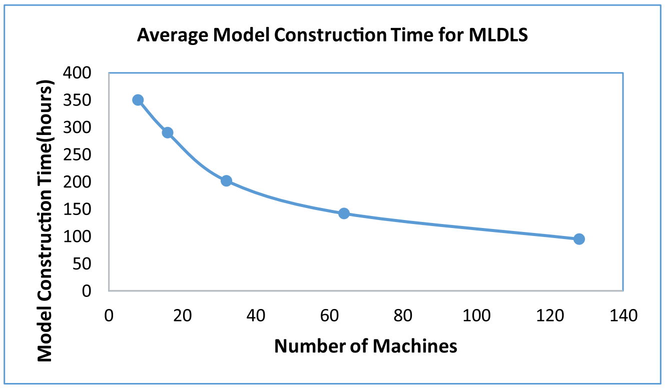

Fig. 15. Average model construction time of MLDLS for 5 × 2 CV F test.

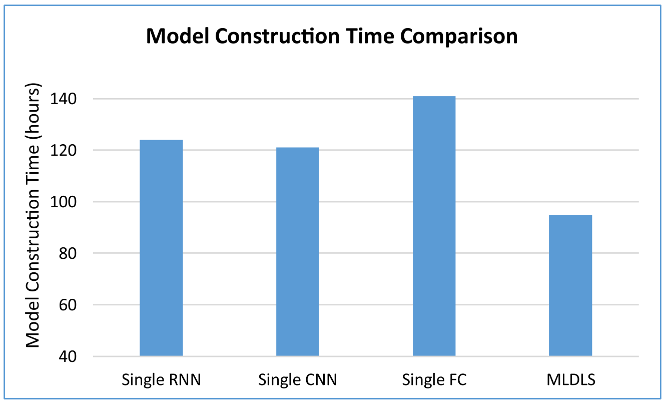

Fig. 16. Comparison of model construction for different deep learning models.

used. The experimental results show that the model construction time for MLDLS has been reduced substantially when multiple machines are deployed. Fig. 16 compares the model construction time for different deep learning models. Experimental results indicate that MLDLS using 128 machines not only reduces the model construction time but also achieves significant detection performance improvement as compared to the single deep learning model.

7. Conclusions and future work

In this paper, we propose a multi-level deep learning system for malware detection by combing different types of deep learning methods in the cluster tree to handle more complex data distributions of malware datasets and enhance the scalability. The main contributions of this work include the following two folds. First, we develop multiple deep learning models in the multi-level tree to learn the complex data distribution of the malware dataset. Second, we utilize the parallel strategy for deep learning model training and selection to reduce the model construction time. Since each deep learning model in the tree structure can focus its efforts to learn the unique data distribution for one group of malware family, our model is more capable to handle the complex malware data distribution. The experimental results demonstrate that our proposed MLDLS improves the performance of malware detection systems as compared to that of the Support Vector Machine, decision tree, the single deep learning model and the ensemble based approach used by other researchers. More accurate detection and less execution time allow us to identify malware more effectively and efficiently. Especially, the significant reduction of model construction time can allow researchers to iterate on machine learning models for malware detection more rapidly and fit larger and more complex deep learning models to the malware dataset (Saxe, & Berlin, 2015).

We will continue and extend our existing work in the following directions. Since experimental results demonstrate that incorporating more training data can improve malware detection accuracy, we plan to include more malware samples into the training data to further boost the performance of malware detection systems. Even though the model construction time is significantly reduced due to parallel strategies, the computational time is still high. In the future, more efficient parallel algorithms will be investigated to further decrease the model construction time.

Conflict of 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.

Credit authorship contribution statement

Wei Zhong: Resources, Validation, Writing - original draft. **Feng Gu:** Validation, Writing - review & editing.

Acknowledgment

This work is partially supported by Research Initiative for Summer Engagement (RISE) from The Office of the Vice President for Research at University of South Carolina.

References

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. F., & Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics, 16*, 412–424.
Benchea, B., & Gavrilut, D. T. (2014). Combining restricted boltzmann machine and one side perceptron for malware detection. *Lecture Notes in Computer Science, 8577*, 93–103.
Chandra, B., & Sharma, R. K. (2017). Deep learning with adaptive learning rate using laplacian score. *Expert Systems with Applications, 63*, 1–7.
Dahl, G. E., Stokes, J. W., Deng, L., & Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*.
Gilbert, D. (2016). *Convolutional neural networks for malware classification* Master thesis.
Gupta, S. K., Rao, K. S., & Bhatnagar, V. (1999). K-means clustering algorithm for categorical attributes. *Lecture Notes in Computer Science, 1676*, 203–208.
Han, J., Kamber, M., & Pei, J. (2011). *Data mining: concepts and techniques* (3rd ed.). Morgan Kaufmann.
Hardy, W., Chen, L., Hou, S., Ye, Y. F., & Li, X. (2016). DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of 12th international conference on Data Mining*.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.
Jung, W., Kim, S., & Choi, S. (2015). Deep learning for zero-day flash malware detection. *Proceedings of 36th IEEE Symposium on Security and Privacy*.
Kazemian, H. B., & Ahmed, S. (2015). Comparisons of machine learning techniques for detecting malicious webpages. *Expert Systems with Applications, 42*(5), 1166–1177.
Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). Deep learning for classification of malware system call sequences. *Lecture Notes in Computer Science, 9992*, 137–149.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of advances in neural information processing systems* (pp. 1097–1105).
Li, Y. C., Nie, X. Q., & Rong, H. (2018). Web spam classification method based on deep belief networks. *Expert Systems with Applications, 96*(15), 261–270.
Masud, M., Khan, L., & Thuraisingham, B. (2011a). *Data mining tools for malware detection*. Auerbach Publications.
Masud, M., Al-Khateeb, T. M., Hamlen, K. W., Gao, J., Khan, L., Han, J. W., et al. (2011b). Cloud-based malware detection for evolving data streasm. *ACM Transactions on Management Information Systems, 2*(3). doi:10.1145/2019618.2019622.
Maxwell, K. (2016). Maltrieve, https://github.com/krmaxwell/maltrieve
McLaughlin, N., Rincon, J. M., Kang, B., Yerma, S., & Ahn, G. J. (2017). Deep android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy* (pp. 301–308).
Obeis, N. T., & Bhaya, W. (2016). Review of data mining techniques for malicious detection. *Research Journal of Applied Science, 11*(10), 942–947.

Offensive Computing. (2017). http://www.offensivecomputing.net/.

Owen, S., Anil, R., Dunning, T., & Friedman, E. (2011). *Mahout in action*. Manning Publications.

Papakostasab, M., & Giannakopoulos, T. (2018). Speech-music discrimination using deep visual feature extractors. *Expert Systems with Applications, 114*, 334–344.

Pascanu, R., Stokes, J. W., Sanossian, H., & Marinescu, M. (2015). Malware classification with recurrent networks. In *Proceedings of IEEE international conference on acoustics, speech and signal processing (ICASSP)*.

Rhode, M., Burnap, P., & Jones, K. (2017). Early stage malware prediction using recurrent neural networks. *Computers & Security, 77*, 578–594.

Roberts, J. M. (2016). Virus Share, https://virusshare.com.

Ronao, C. A., & Cho, S. B. (2016). Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications, 59*, 235–244.

Saxe, J. (2017). Invincea Labs.

Saxe, J., & Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. In Proceedings of 10th international conference on malicious and unwanted software (MALWARE).

Shahzad, R. K., & Lavesson, N. (2013). Comparative analysis of voting schemes for ensemble-based malware detection. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 4*(1), 98–117.

Sung, A., Xu, J., Chavez, P., & Mukkamala, S. (2014). Static analyzer of vicious executables (save). In *Proceedings of the 20th ACSAC* (pp. 326–334).

Tobiyama, S., Yamaguchi, Y., Shimada, Y. H., Ikuse, T., & Yagi, T. (2014). Malware detection with deep neural network using process behavior. In *Proceedings of IEEE 40th annual computer software and applications conference* (pp. 577–582).

Turki, T. O., & Bhaya, W. (2016). Review of data mining techniques for malicious detection. *Research Journal of Applied Sciences, 11*(10), 942–947.

VX Heaven virus collection. (2016). http://vxer.org/.

VirusSign. (2017). http://www.virussign.com/.

Wei, J., hua, J., Chen, K., Zhou, Y., & Tang, Z. (2017). Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications, 69*, 29–39.

Ye, Y. F., Li, T., Adjeroh, D., & Iyengar, S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys, 50*(3), 40.

Yue, S. Q. (2017). *Imbalanced Malware images classification: A CNN based approach*. Cornell University Library.

Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Science and Technology, 21*(1), 114–123.