Computers & Security 89 (2020) 101655

Contents lists available at ScienceDirect

Computers & Security

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

An improved two-hidden-layer extreme learning machine for malware hunting

Amir Namavar Jahromi[a], Sattar Hashemi[a], Ali Dehghantanha[b], Kim-Kwang Raymond Choo[c,*], Hadis Karimipour[d], David Ellis Newton[e], Reza M. Parizi[f]

[a] Machine Learning Lab, Department of Computer Science & Engineering & Information Technology, Shiraz University, Shiraz, Iran
[b] Cyber Science Lab, School of Computer Science, University of Guelph, Ontario, Canada
[c] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249, USA
[d] Smart Cyber-Physical System Lab, School of Engineering, University of Guelph, Ontario, Canada
[e] School of Computer Science, University of Salford, Manchester, UK
[f] Department of Software Engineering and Game Development, Kennesaw State University, Marietta, GA 30060, USA

A R T I C L E   I N F O

Article history:
Received 15 March 2019
Revised 1 October 2019
Accepted 22 October 2019
Available online 21 November 2019

Keywords:
Extreme learning machine (ELM)
Ensemble learning
Deep learning
Stacked long short term memory (LSTM)
Convolutional neural network (CNN)
Threat hunting
Malware forensics

A B S T R A C T

Detecting unknown malware and their variants remains both an operational challenge and a research challenge. In recent years, there have been attempts to design machine learning techniques to increase the success of existing automated malware detection and analysis. In this paper, we build a modified Two-hidden-layered Extreme Learning Machine (TELM), which uses the dependency of malware sequence elements in addition to having the advantage of avoiding backpropagation when training neural networks. We achieve this goal by using partially connected networks between the input and the first hidden layer. These are then aggregated with a fully connected network in the second layer. Finally, we utilize an ensemble to improve the accuracy and robustness of the system for malware threat hunting. The proposed method speeds up the training and detection steps of malware hunting, in comparison to stacked Long Short Term Memory (LSTM) and Convolutional Neural Network (CNN). Specifically, this is achieved by avoiding the backpropagation method and using a more simple architecture. Hence, the complexity of our final method is reduced, which leads to better accuracy, higher Matthews Correlation Coefficients (MCC), and Area Under the Curve (AUC), in comparison to a standard LSTM with reduced detection time. Our proposed method is especially useful for malware threat hunting in safety-critical systems, such as electronic health or Internet of Battlefield / Military of Things, since the enormous size of the training data makes it impractical to use complex models (e.g., deep neural networks). In addition in safety-critical systems, both training and detection speeds, as well as the detection rate, are equally important. Our research results in a powerful network that can be used for all platforms with a range of malware analysis. The proposed approach is tested on Windows, Ransomware, Internet of Things (IoT) and a mix of different malware samples datasets. For example, our evaluation using an IoT-specific dataset reports an accuracy of 99.65% in detecting IoT malware samples with an AUC of 0.99, and an MCC of 0.992; thus, outperforming standard LSTM based methods for IoT malware detection in all metrics.

1. Introduction

A malicious software (also known as malware) is a program that is designed to disrupt, damage or gain unauthorized access to computer system(s) or computing device(s). For example, more than 239 million new malware samples were reportedly detected in the 3rd quarter of 2018 (Chebyshev et al., 2018), and more than 669 million new malware variants were detected in 2017 (an increase of 80.1% from the prior year) (Symantec, 2018). Also, findings released by another security company indicated that the number of Internet of Things (IoT) malware increased nine-fold in the third quarter of 2018, compared to the last quarter of 2017 (McAfee, 2018). Hence, it is clear that malware remains a serious threat to our national cyber infrastructure and IoT services (Azmoodeh et al., 2018; Dovom et al., 2019; Karimipour et al., 2019). In recent times, there have been attempts to utilize artificial

* Corresponding author.
E-mail addresses: a.namavar@aut.ac.ir (A. Namavar Jahromi), s_hashemi@shirazu.ac.ir (S. Hashemi), Ali@cybersciencelab.org (A. Dehghantanha), raymond.choo@fulbrightmail.org (K.-K.R. Choo), hkarimi@uoguelph.ca (H. Karimipour), D.E.Newton@salford.ac.uk (D.E. Newton), rparizi1@kennesaw.edu (R.M. Parizi).

https://doi.org/10.1016/j.cose.2019.101655
0167-4048/© 2019 Elsevier Ltd. All rights reserved.

intelligence (broadly defined to include machine and deep learning techniques) to detect malware samples more efficiently.

Generally, there are two main categories of machine learning based malware detection techniques, including signature-based detection techniques (Kephart and Arnold, 1994; Preda et al., 2008; Santos et al., 2009; Ye et al., 2017). Software products of conventional anti-malware companies (e.g., Kaspersky, Bitdefender, and Symantec) use this approach for their malware detection. A signature is a short sequence of bytes unique to each known malware, which is used for malware detection with a small rate of error (Ye et al., 2011). These techniques usually fail to detect new threats, thereby leaving them undetected for a prolonged period (Ye et al., 2017). As observed by Damballa (2008), 15% of malware samples are still undetected after 180 days. In other words, such approaches are not capable of detecting all malware variants. To overcome the drawbacks of signature-based methods and increase the efficiency of malware detection, heuristic malware detection techniques (e.g., classification, image processing, and natural language processing (NLP)) were proposed in the late 1990s (Brunnstein et al., 1991; Shalaginov et al., 2018; Tesauro et al., 1996; Ye et al., 2017).

Earlier generations of heuristic-based malware detection approaches generally convert a malware sample to a basic form (e.g. image, signal or text) prior to extracting the relevant features for further analysis (Farrokhmanesh and Hamzeh, 2016; Fu et al., 2018; Hashemi and Hamzeh, 2018; Mohammadi et al., 2019; Santos et al., 2010; Zhang et al., 2016). While such approaches may achieve a reasonable accuracy, the time required to change the sample representation (static or dynamic like OpCodes, Bytecodes, library calls, etc.) and also the time taken for feature extraction (1.3 to 3 s per sample for pre-processing in the approaches of Hashemi and Hamzeh, 2018; Karbab et al., 2018; Ni et al., 2018; Zhang et al., 2016 for instance), resulted in a poor performance on both the training and testing phases of the algorithms (Baldwin and Dehghantanha, 2018). Thus, a number of researchers have attempted to use NLP techniques on malware static features (like Opcode and file sections) to identify malicious programs (Kang et al., 2016; Xu et al., 2016). Although NLP techniques can help to reduce the processing time (by conserving the raw representation), they resulted in low detection accuracy (below 87%) due to improper features extraction (like n-grams, tf-idf, etc.).

Convolutional Neural Network (CNN), a class of deep learning algorithms, is a potential approach that can be leveraged to improve the detection accuracy, ranging from 92% to 99% in the detection of malware using a range of dynamic and static features according to Ni et al. (2018), Karbab et al. (2018), Yakura et al. (2018), Kabanga and Kim (2018), Cui et al. (2018), Meng et al. (2017), McLaughlin et al. (2017) and Le et al. (2018). However, CNN-based approaches require longer pre-processing time and significant training data; thus, necessitating the deployment of powerful hardware such as graphics processing units (GPUs) (Goodfellow et al., 2016). As a key assumption of CNN-based approaches is independent and identical distribution in the input data, their accuracy drops significantly for variable-length sequence of malware OpCodes or samples with interdependent library calls as input (Athiwaratkun and Stokes, 2017; Goodfellow et al., 2016; Nauman et al., 2018).

To address the above limitations, Recurrent Neural Network (RNN) techniques can be used for variable-length sequential data in deep neural networks, and original malware features such as OpCodes (Hochreiter and Schmidhuber, 1997). RNN is a multilayer neural network with tied weights where each layer may receive two inputs, namely: one from the original input data and the other from the previous layer. As layers can be generated dynamically, RNN-based techniques are capable of processing variable-length input data (Schmidhuber, 2015). However, when the length of the input data increases, traditional RNN networks with backpropagation suffer from the vanishing gradient problem (Hochreiter and Schmidhuber, 1997), which causes insufficient learning in initial layers. This limits the utilization of RNN techniques to malware samples with a relatively short length of features (i.e., a short sequence of OpCodes or a limited number of library calls). Thus, while maintaining the power of deep networks, stacked Long Short-Term Memory (LSTM) networks can be considered (Hochreiter and Schmidhuber, 1997). Specifically, a stacked LSTM uses a hierarchy of LSTM networks to map an input sequence into another space with an automated feature learning procedure. Stacked LSTM based approaches for malware detection can deal with almost any length of sequential input data and offer a high accuracy (e.g., between 97% and 99.7% reported by Homayoun et al. (2018), Yan et al. (2018), HaddadPajouh et al. (2018), Xiao et al. (2017), Xu et al. (2018), Vinayakumar et al. (2018) and Yuan et al. (2016)).

While approaches based on deep neural networks generally achieve relatively high malware classification rate, their training phase is lengthy and power-consuming (Huang et al., 2006). Hence for large datasets, such approaches are impractical, particularly as the number of malware samples increases. Hence, more efficient approaches are required. Huang et al. (2004), for example, proposed a method to avoid the backpropagation of neural networks, which is coined Extreme Learning Machine (ELM). ELM is a Single hidden Layer Feedforward Network (SLFN) with random weights of input to the hidden layer and biases (Huang et al., 2006). Compared to conventional parameter updating techniques, ELM is known to achieve extremely fast learning speed, excellent generalization capability, more straightforward implementation, and less human intervention characteristics (Cao et al., 2016a; Huang, 2015). However, it is pointed out that in specific applications, ELM may have poor performance in handling noisy data, such as in image recognition (Cao et al., 2016b; 2015). Also, its predictive performance is still affected by the input weights of the neural network and the bias of the hidden layer (Li et al., 2018; Juan Lu et al., 2014). Due to the input weights of the neural network and the bias of hidden layers random initialization, the performance of ELM is not versatile when implemented with consistent training and test samples. In other words, the stability of ELM is not ideal (Li et al., 2018). However, the stability problem of the ELM can potentially be mitigated using ensemble methods (Cao et al., 2016a; Li et al., 2018; Liu and Wang, 2010; Juan Lu et al., 2014).

Attempts in using ELM for malware detection, such as those of Zhang et al. (2015), Demertzis and Iliadis (2016), Sun et al. (2017) and Yousefi-Azar et al. (2018a), have known to achieve reasonable accuracy and training speeds based on their evaluations using various datasets. However, these approaches generally used pre-processing techniques to extract Independent and Identically Distributed (iid) features from malware samples and then used an ELM model for classification. This, however, incurs additional time for pre-training and also does not consider relations between the sample's original features. Therefore, we propose a new architecture for ELM network to facilitate malware detection, so that we can avoid having a pre-training step while considering short-term dependencies between the sample's features.

A summary of the contributions of this paper is as follows.

(1) To mitigate existing ELM-based approaches' limitation of not being able to handle sequential data while taking into account identically distributed features, we propose a two-hidden-layer ELM-based technique that is capable of capturing the dependencies between adjacent features. This eliminates the need for feature extraction, and original bytecodes, Opcodes, or system-calls can be fed to the model directly. This model considers local dependencies of adjacent features

in the first layer and obtains the global dependencies of the model in the second layer.

(2) As ELM-based approaches use random weights and biases without relying on any iterative learning, their results vary in different runs based on their random initialization. Therefore, we propose a bagging-based ensemble method to stabilize the model. To achieve this goal, we train 30 different classifiers with random feature numbers and window sizes and then choose the best ten classifiers for our voting procedure. When our model sees the test (unseen) sample, this sample is passed to all selected classifiers and a voting mechanism is initialized to select the best class based on the classifier's output.

(3) Since the proposed method has a simpler architecture than other powerful methods like deep neural networks, it speeds up the training and detection time of the model while the performance of the model (for various metrics like ACC, AUC, MCC) is comparable to those of stacked LSTM (as a deep recurrent neural network) and CNN.

We also remark that our proposed approach is particularly useful for malware threat hunting in safety critical systems, such as in adversarial settings (e.g., military or battlefield settings)), where the very large training data size complicates the use of complex models such as deep neural networks.

The rest of the paper is organized as follows. In Sections 2 and 3, we present background materials relating to ELM and TELM and existing literature on ELM-based malware detection approaches, respectively. Our proposed approach is then presented in Section 4, and the datasets and experimental setup for the evaluation of the approach described in Section 5. Specifically, the proposed approach is evaluated using several malware datasets namely: VXHeaven dataset (VXHeaven, 0000), Kaggle dataset (Microsoft, 2015), Windows ransomware dataset (Homayoun et al., 2017), IoT malware dataset (HaddadPajouh et al., 2018), and a combined ransomware and IoT malware samples. A comparative summary of the performance for the proposed approach, the stacked LSTM with 4 hidden layers, three CNN models, the original ELM (Huang et al., 2004), and two-hidden-layer ELM (TELM) (Qu et al., 2015) is then presented in Section 6. From the findings, one can observe that the proposed approach outperforms all CNN models in terms of accuracy and MCC for most of the datasets (except for the VXHeaven and Kaggle datasets). The proposed approach also outperforms ELM and TELM in all metrics. Moreover, the detection time of the proposed approach is 1.2 ms and its training time is 2.343 s; in other words, the proposed approach is faster than stacked LSTM with 739.672 s of training time and 2.7 millisecond detection time. Also, the proposed method is faster than two-layer CNN with 418.36 s of training and 2.2 ms detection time. Sections 7 and 8 respectively present the discussion and conclusion.

2. Background

As previously discussed before, the ELM network (see Fig. 1) can be used to avoid the need for backpropagation algorithms in neural networks and to make these networks faster and more general (Huang et al., 2004; 2006). The original ELM was introduced in 2004 (Huang et al., 2004) for SLFNs with random input to hidden layer weights. Consider N arbitrary distinct samples $(x_i, y_i)(i = 1, 2, \cdots, N)$ where x_i is an input sample with $[x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \Re^n$ features and y_i is its output (e.g. labels) that can have more than one dimension (e.g. regression), where $y_i = [y_{i1}, y_{i2}, \cdots, y_{im}]^T \in \Re^m$. Inputs and outputs are shown in a matrix format where $X = [x_1, x_2, \cdots, x_N]^T$ and $Y = [y_1, y_2, \cdots, y_N]^T$ illustrate the input and output matrices, respectively, and T indicates matrix/vector transposition. Suppose that we have L neurons in the hidden layer and that the activation function of the hidden layer is $g(x)$. ELM selects the weight matrix of inputs to the hidden layer $W = [W_1, W_2, \cdots, W_L]^T \in \Re^{L\times n}$ randomly that connects the input layer to the hidden layer. Biases of the hidden layer $B = [b_1, b_2, \cdots, b_L] \in \Re^{L\times N}$ are selected randomly; noting that the B and W matrices remain unchanged during the training phase. Hence, after setting W and B, the value of the hidden layer H can be calculated using Eq. (1) below.

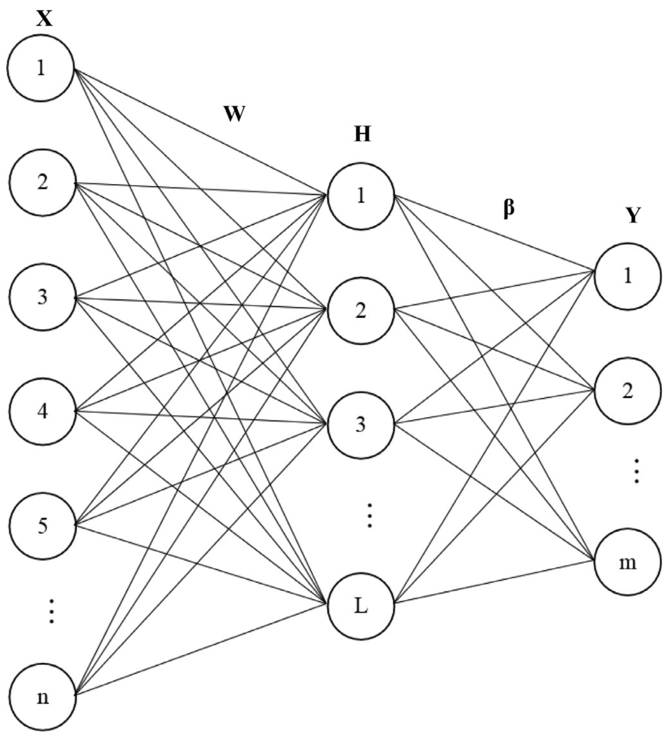

Fig. 1. Architecture of ELM model.

$$H = g(WX + B) \in \Re^{N\times L} \tag{1}$$

In a neural network such as Fig. 1, the output can be calculated by Eq. (2):

$$Y = H\beta \tag{2}$$

where $\beta = [\beta_1, \beta_2, \ldots, \beta_L]^T \in \Re^{L\times m}$ is the weight matrix between the hidden layer H and the output layer Y.

The only parameter that should be calculated in the ELM method is the β matrix which is calculated by the least-square method as shown in Eq. (3):

$$\beta = H^{\dagger}Y \tag{3}$$

where $H\dagger$ is the Moore-Penrose (MP) generalization inverse of a matrix H that is calculated using Eq. (4):

$$H^{\dagger} = \begin{cases} (H^TH)^{-1}H^T, ifH^THisnonsingular \\ H^T(HH^T)^{-1}, ifHH^Tisnonsingular \end{cases} \tag{4}$$

Note that the ELM is not an iterative method and the β parameter is only calculated once. After calculating β, the training phase is ended and, for each test sample x, the output can be calculated using Eq. (5):

$$Y = g(Wx + B)\beta \tag{5}$$

Two-hidden-layer Extreme Learning Machine (TELM) is an extension on the ELM with two hidden layers that was presented in 2015 by Qu et al. (2015). In this model, the size of the hidden layers should be equal and both have L neurons. Fig. 2 (with solid

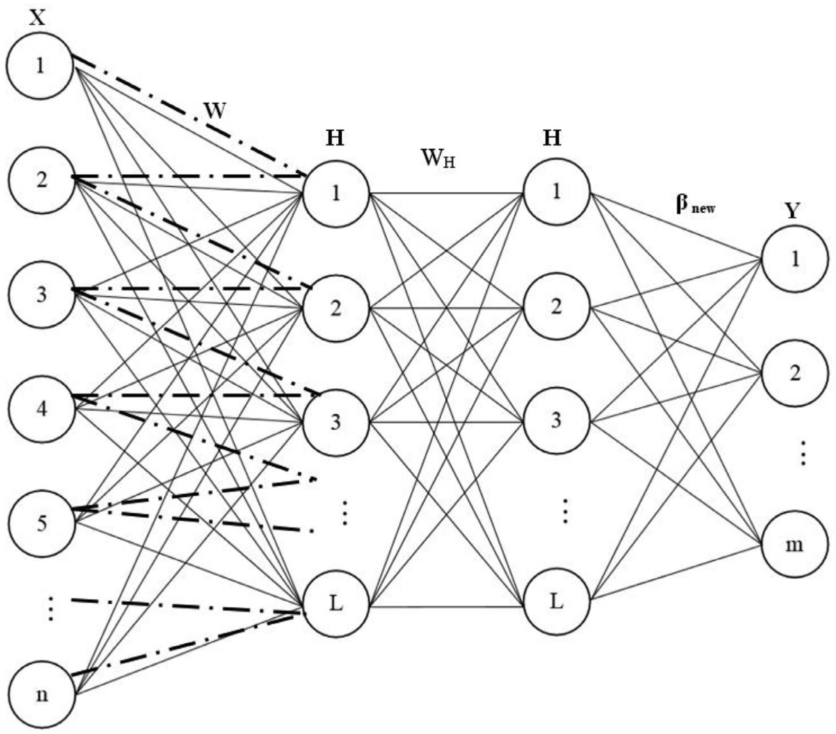

Fig. 2. The solid lines between the input and the first hidden layer show original TELM model while the dashed line connections illustrate the proposed method.

lines between the input and the first hidden layer) below shows the TELM model.

At first, the network is considered as an SLFN (by removing the second hidden layer and weights between the first and second hidden layers), setting the W and B matrices randomly and calculating β using Eqs. (1), (2) and (3). After that, we can calculate the expected output of the second hidden layer shown in Eq. (6):

$$\hat{H}_2 = Y\beta^{\dagger} \tag{6}$$

where $\hat{H}_2$ is the expected output of the second hidden layer of the network, Y is the output matrix/vector and $\beta\dagger$ is the MP generalized inverse of β. By having $\hat{H}_2$, we can calculate the weights between the first and second hidden layers through using Eq. (7):

$$W_{HE} = g^{-1}(\hat{H}_2)H_E^{\dagger} \tag{7}$$

where $W_{HE} = [B_1 \;\; W_H]$ with B_1 being the bias matrix of the second hidden layer and W_H is the weight matrix between the first and second hidden layers, g^{-1} is the inverse of the second hidden layer's activation function, $\hat{H}_2$ is the expected output of the second hidden layer and $H_E^{\dagger}$ is the MP generalized inverse of $H_E = [1 \;\; H]^T$. Now the actual output of the second hidden layer can be calculated using Eq. (8):

$$H_2 = g(W_{HE}H_E) \tag{8}$$

Finally, the weight matrix β_{new} between the second hidden layer and the output layer can be calculated us9ing Eq. (9):

$$\beta_{new} = H_2^{\dagger}Y \tag{9}$$

where $H_2^{\dagger}Y$ is the MP generalized inverse of H_2. At the testing phase for each test sample x, the output of the network is then calculated using Eq. (10):

$$Y = g\{[W_H g(Wx + B) + B_1]\}\beta_{new} \tag{10}$$

Fig. 3 describes the algorithm of the TELM.

```
1: Input: $N$ training samples $X = [x_1, x_2, ..., x_N]^T$, $Y = [y_1, y_2, ..., y_N]^T$,
   and $2L$ hidden neurons in total with activation function $g(x)$
2: Randomly generate the connection weight matrix between the input layer
   and the first hidden layer $W$ and the $B$ matrix of the first hidden layer $B$
   and for simplicity, $W_{IE}$ is defined as $[B \;\; W]$ and similarly, $X_E$ is defined
   as $[1 \;\; X]^T$.
3: Calculate $H = g(W_{IE}X_E)$
4: Obtain weight matrix between the second hidden layer and the output layer
   $\beta = H^{\dagger}Y$
5: Calculate the expected output of the second hidden layer $H_1 = Y\beta^{\dagger}$
6: Determine the parameters of the second hidden layer (connection weight
   matrix between the first and second hidden layer and the bias of the second
   hidden layer) $W_{HE} = g^{-1}(H_1)H_2{}^{\dagger}$
7: Obtain the actual output of the second hidden layer $H_2 = g(W_{HE}H_E)$
8: Recalculate the weight matrix between the second hidden layer and the
   output layer $\beta_{new} = H_2{}^{\dagger}Y$
9: Output: The final output of TELM is $f(x) = g\{[W_H g(WX + B) +
   B_1]\}\beta_{new}$
```

Fig. 3. Algorithm of TELM.

3. Related work

To speed up the learning of artificial neural networks and gain a better generalization, some researchers utilized ELM for malware detection (Demertzis and Iliadis, 2016; Kozik, 2018; Sun et al., 2017; Yousefi-Azar et al., 2018a; 2018b; Zhang et al., 2015). For example, in Zhang et al. (2015), Dalvik instructions (Wognsen et al., 2014) were firstly extracted from bytecode as features. After making vectors from these extracted features, PCA (Smith, 2002), KLT (Rao and Yip, 2001), and ICA (Hyvärinen and Oja, 2000) dimension reduction algorithms were carried out, and one ELM was fitted on each model. Also, they stacked the previous models and tested the stacked model with and without the feature extraction for malware detection. They reported the accuracy between 92.10% and 97.53% where the lower accuracy belonged to the ELM model on PCA features, and the highest one gathered using the stacked model with feature extraction. Demertzis and Iliadis (2016) used hardware information (CPU, memory, battery, network, etc.) and application permissions as the features for an Android malware detection system. They used ELM and Evolving Spiking Neural Networks (eSSNs) (Schliebs and Kasabov, 2013) methods as their models and obtained an accuracy between 71.27% and 89.47% for ELM and between 79.30% and 98.20% for the eSSN method. Sun et al. (2017) used an ELM method for Android malware detection using static features of permission and API calls and reported an accuracy of 95%. Also, Yousefi-Azar et al. (2018a) looked at the malware samples as text and hashed the byte n-grams to obtain a lower dimension and classified this new representation with various algorithms such as ELM, SVM, KNN, and XGBoost. They reported an accuracy between 92.34% (for SVM) and 97.31% (for ELM). Also, they proposed a method in which they used a two-layer neural network for malware detection. The first hidden layer is responsible for extracting the tf-simhashing features, while the second one is the ELM layer. They tested their method on several datasets and gained accuracy between 91.22% and 99.67% (Yousefi-Azar et al., 2018b). Also, Kozik (2018) proposed a distributed and cost-sensitive ELM method for malware detection based on their network traffic activity. He reported an accuracy between 76% and 95% on several scenarios.

As it can be seen in all the previous works, researchers extracted features from malware data rather than working directly with native malware features such as OpCodes, bytecodes, and system calls. Moreover, ELM methods were introduced to speed up artificial neural networks by using least square methods instead of the backpropagation method (Huang et al., 2004; 2006). However, these methods and their variants are not suitable for malware detection since they need their input data independent and identically distributed. Hence malware researchers had to pre-process malware to extract identically distributed features from samples and then pass them to the ELM model.

4. Proposed method

In this paper, we introduced a new ELM that considers dependencies between original malware features (like OpCodes, system calls, etc.) and can train the model without any pre-processing. We consider an ELM model with two hidden layers such as TELM but with different connections of input to the first hidden layer. To achieve this, we use a window on features that defined the size of dependent features. To distinguish them from other features, the first layer of the network is partially connected, and each window members are connected to a single neuron. Fig. 2 (with dashed lines between the input and the first hidden layer) shows the architecture of our proposed method.

As illustrated in Fig. 2 (dashed connections), input features (1 to n) are categorized in groups with k features (in this figure $k = 2$) in which all the group members are connected to a unique neuron. So the intermediate neurons (H) keep the information of adjacent features and their local dependencies. The second hidden layer (H_2) is responsible for aggregating the local dependencies of the previous layer to achieve the global dependencies of the features. After that, the output of the model can be calculated by using the global representation and output weights (β).

Consider that we have N distinct samples $(x_i, y_i)(i = 1, 2, \cdots, N)$ where x_i is the input sample with $[x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \Re^n$ features and y_i is its output (e.g. labels) that can have more than one dimension (e.g. regression), $y_i = [y_{i1}, y_{i2}, \cdots, y_{im}]^T \in \Re^m$. In the initial step, we consider window sizes K where $1 \leq K \leq n$. 1 and n means that the features are independent and highly dependent, repectively. The number of neurons of hidden layers is equal to $L = n - K + 1$.

In this first step, we categorize the input features into L groups $X^j = [x_1^j, x_2^j, \cdots, x_N^j]$ where $1 \leq j \leq L$ and $x_i^j = [x_i^j, \cdots, x_i^{j+K}]^T$ is a feature vector of the i-th sample with K features (j^{th} to $(j+k)^{th}$ features). So $X^j \in \Re^{N \times K}$. The weight matrix of $W = [w^1, w^2, \cdots, w^L]$ is selected randomly with L weight vectors of $K \times 1$ that W_j is the weights of j^{th} group of features. The output of the first hidden layer is $H = [h^1, h^2, \cdots, h^L] \in \Re^{N \times L}$ and h^j is calculated by Eq. (11) below.

$$h^j = g(w^j X^j) + B \tag{11}$$

After calculating the output of this first layer, like TELM, we consider our network to be a one hidden layer network and calculate the output weights β using Eqs. (12) and (13):

$$Y = H\beta \tag{12}$$

$$\beta = H^{\dagger} Y \tag{13}$$

After that, we can calculate the expected output of the second hidden layer using Eq. (14):

$$\hat{H_2} = Y\beta^{\dagger} \tag{14}$$

where $\hat{H_2}$ is the expected output of the second hidden layer of the network, Y is the output vector, and $\beta\dagger$ is the MP generalized inverse of β. By having $\hat{H_2}$ we can calculate the weights between the first and second hidden layers by Eq. (15):

$$W_{HE} = g^{-1}(\hat{H_2}) H_E{}^{\dagger} \tag{15}$$

where $W_{HE} = [B_1 W_H]$ and B_1 is the bias matrix of the second hidden layer and W_H is the weight matrix between the first and second hidden layers, g^{-1} is the inverse of the second hidden layer's activation function, $\hat{H_2}$ is the expected output of the second hidden layer, and $H_E\dagger$ is the MP generalized inverse of $H_E = [1 \ \ H]^T$. We chose the sigmoid function as the activation function (g) for both layers So, g^{-1} is a logit function. Now the actual output of the second hidden layer can be calculated using Eq. (16):

$$H_2 = g(W_{HE} H_E) \tag{16}$$

Finally, the weight matrix β_{new} between the second hidden layer and output layer can be calculated using Eq. (17):

$$\beta_{new} = H_2{}^{\dagger} Y \tag{17}$$

where $H_2\dagger$ is the MP generalized inverse of H_2. At the testing phase for each test sample x, the output of the network can be calculated using Eq. (18):

$$Y = g\{[W_H g(Wx + B) + B_1]\}\beta_{new} \tag{18}$$

Based on the literature (Alom et al., 2016; Cao et al., 2016a; Huang et al., 2011; Li et al., 2018; Liu and Wang, 2010; Wang et al., 2011) and our observations, one of the problems associated with the ELM based methods is the randomness of the results. This is due to the random weights' matrix which affects the output of the model and makes the results unstable. To illustrate, it causes the accuracy to fluctuate between 73% and 100% for the IoT dataset.

To solve this problem, we propose an ensemble model where 30 modified TELM models are trained with random input (number of features) and window sizes. Based on their accuracy on training data, the top 10 models are selected as our final models, and the final decision is made by majority voting between them. The class that was selected by the majority number of classifiers was selected as the final label. The number of selected models is achieved following several tests after comparing the accuracy and MCC of the result. To have a fair comparison, this method is repeated for ELM and original TELM as well.

Table 1
Malware families and number of their samples in Kaggle dataset.

#	Family	Number of samples
1	Gatak	1219
2	Kelihos V1	392
3	Kelihos V3	2938
4	Lollipop	2476
5	Obfuscator	1013
6	Ramnit	1541
7	Simda	42
8	Traur	751
9	Vundo	453

5. Data preparation and experimental setup

We utilized three Windows malware samples datasets including VXHeaven dataset (VXHeaven), Microsoft Kaggel dataset (Microsoft, 2015) and a ransomware dataset (Homayoun et al., 2017) to evaluate our classification model. The first dataset contained 3300 samples where 2200 of them were used for training, and 1100 samples were used for testing. The dataset is balanced, and the samples were taken randomly from the entire VX-Heaven dataset and were labeled as malware. Benign samples were collected from Windows and other known programs. Benign files were checked using two commercial anti-viruses to ensure that they are not malicious. Samples of Kaggle dataset were labeled as nine families of malware variants without any benign samples. Table 1 shows the number of samples in each family of this dataset. This dataset contained 10,825 samples that were analyzed statically to extract their opcodes. Our last Windows malware dataset contained system calls of 1801 samples that were categorized into ransomware and benign samples for binary classification by Homayoun et al. (2017). Also, we used an IoT dataset (HaddadPajouh et al., 2018) which contained a total of 552 samples from which 271 samples were labeled as benign and 281 samples were labeled as malware. Finally, we made a bigger dataset containing samples of both ransomware and IoT datasets as our third dataset and called it IoT+Ransomware dataset.

We tested binary malware detection techniques on these datasets (except the Kaggle dataset) with malware and benign labels. For Kaggle dataset, which has more than two classes, we built a classification model to identify malware families.

Table 2 summarizes the basic information and characteristics of all datasets that we used in this research.

We ran all our experiments on a Core i7-4500U laptop with 8 GB of RAM and Windows 10 operating system. Samples were categorized in training and test groups with 80% for training and others for testing. We used this several times to gain reliable results. To pass the input data to our model, no pre-processing is needed and raw feature vectors can enter our model directly. So, the proposed model is faster than others that need some pre-processing for feature extraction (Demertzis and Iliadis, 2016; Sun et al., 2017; Yousefi-Azar et al., 2018a; Zhang et al., 2015).

Basic metrics for evaluating the performance of machine learning algorithms generally include True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), as shown in Eqs. (19)–(22), respectively.

$$TP = \sum samples\ correctly\ classified\ as\ malware \tag{19}$$

$$TN = \sum samples\ correctly\ classified\ as\ benign \tag{20}$$

$$FP = \sum samples\ wrongly\ classified\ as\ malware \tag{21}$$

$$FN = \sum samples\ wrongly\ classified\ as\ benign \tag{22}$$

Using the above metrics, we can define the True Positive Rate (TPR), False Positive Rate (FPR), Accuracy (ACC), Matthews Correlation Coefficients (MCC), the Receiver Operating Characteristics (ROC) curve, and Area Under Curve (AUC) to measure the performance of machine learning algorithms in performing malware detection.

- **TPR** measures the proportion of actual positives that are correctly identified as positive using the classification model. In our experiments, the percentage of malware samples that are correctly classified as malware are calculated using Eq. (23).
- **FPR** is calculated as the ratio between the number of negative events wrongly categorized as positive, and the total number of actual negative events (see Eq. (24)).
- **ACC** indicates the number of samples that were classified correctly over the entire dataset (see Eq. (25)).
- **MCC** is a measure of the quality of binary classification. The MCC is a correlation coefficient between the observed and predicted binary classifications, and returns a value between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 no better than the random prediction, and -1 indicates total disagreement between prediction and observation (see Eq. (26)).
- **ROC** is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
- **AUC** is the probability that a classifier ranks a randomly chosen positive instance (i.e., a malware) higher than a randomly chosen negative one (i.e., a benign application) (Fawcett, 2006). An AUC value of 0 means that the algorithm classifies all samples wrongly, 0.5 means that the algorithm did not work better than random classifier, and 1 means that it can classify all the samples correctly.

$$TPR = \frac{TP}{TP + FN} \tag{23}$$

$$FPR = \frac{FP}{TN + FP} \tag{24}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{25}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{26}$$

In this paper, accuracy, MCC, AUC, and ROC were used to evaluate the performance of the proposed model.

6. Results

As mentioned before, each dataset contains sequences of malware and benign samples including opcodes and system calls. We compared the performance of our model to the standard stacked LSTM, CNN, ELM and original TELM using accuracy, AUC and MCC metrics.

LSTM was selected since it is a popular deep recurrent neural network that considers both short and long term dependencies of the sequence elements. The used LSTM contains four-hidden layers, as shown in Fig. 4. In this network, the first 3 layers were responsible for changing the input representation to another one, and the last hidden layer was responsible for converting this sequence

Table 2
Dataset description.

Dataset	Type of analysis	Type of data	# of Classes	# of Instances
Ransomware	Dynamic	System Call	2	1801
IoT	Static	Opcode	2	552
VXHeaven	Static	Opcode	2	3300
Kaggle	Static	Opcode	9	10825
IoT+Ransomware	Dynamic, static	System Calls, Opcode	2	2353

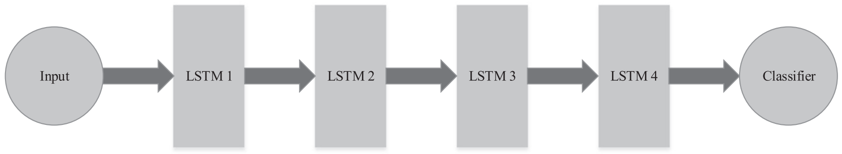

Fig. 4. Architecture of stacked LSTM used for comparison.

shaped representation to a vector representation suitable for classification. The last component of this network was a softmax classifier. The proposed method was compared with three CNN models, namely: a one-layer model, a two-layer model, and a three-layer model. First, the step of training 30 modified TELM was not considered, and a one-layer CNN was built in which the convolutional layer had ten filters (similar to the selected ten modified TELM models in the proposed method), and the pooling layer is responsible for aggregating the results of filters (similar to the voting procedure in the proposed method). A two-layer CNN was used for comparison, where the first layer had 30 filters, and the second layer had ten filters similar to the proposed model with 30 and ten ensembles, respectively. This model utilizes a pooling layer to achieve a single result from all filters. It also uses a three-layer CNN model to observe the changes with different depths.

As AUC and MCC are binary metrics, the one vs. all technique is used to evaluate these metrics for the Kaggle dataset. In this technique, samples of one class are considered as positive and others are considered as negative and the model is trained and evaluated accordingly. This procedure is repeated for all the classes and the result is the average of the one vs. all models' results.

As can be seen in Table 3, the proposed method achieved better accuracy, AUC and MCC in comparison with the standard stacked LSTM, ELM and original TELM in all the datasets except Kaggle.

It can be seen from Table 3 that our proposed method is not only an improvement on deep network and related ELM-based methods in all ACC, AUC and MCC metrics, but also being more stable than ELM, TELM and other stacked LSTM methods. Also, the proposed method outperforms the CNN in accuracy and MCC with almost all datasets, but the CNN achieved better AUC in almost all datasets.

To visually show the comparison between ELM based methods, the ROC curve of the proposed method, ELM, and TELM for ransomware dataset, IoT dataset, IoT+ransomware, and VXHeaven datasets are shown in Figs. 5–8, respectively.

In addition to these metrics, we compared the training and detection time of our proposed method to the other methods in Table 4. These times are for IoT+ransomware datasets.

7. Discussion

To show the power of our proposed method, we compared it with three alternative methods. First, we compared our model with stacked LSTM as a popular deep neural network method. Stacked LSTM is a powerful machine learning model for sequential data and time series (Athiwaratkun and Stokes, 2017; Nauman et al., 2018). In addition, we compared our model to two ELM-based methods, ELM and TELM. Also, we compare our method with a two-layer CNN. As shown in Table 3, our proposed method outperforms all these models in all datasets with all metrics. Also, our experiments and Figs. 5–7 show the superior robustness of our model against other comparable ELM-based methods as well as the AUC metric of 0.59 and 0.83 for TELM on two runs. The IoT+Ransomware datasets in Fig. 7 have about 4% difference in ACC and more than 0.1 on the MCC metric. The same conclusion can be seen in Figs. 5 and 6 for both TELM and ELM methods on ransomware and IoT datasets, respectively.

Figs. 5–7 show the results of ROC and AUC of our method compared to ELM and TELM. As seen in these figures, our method has a lower false negative rate with higher true positive rate. Thus, considering the locality of input features in ELM leads to better detection of malware samples with lesser false detection.

To show the speed of training and detection time, we compared the proposed model to the stacked LSTM as a deep neural network method in Table 3. As shown in Table 3 the stacked LSTM achieved a better ACC but lower AUC and MCC when compared to our model with Kaggle dataset. This is because this dataset is a multi-class and ACC is not a good metric for evaluating such a multi-class classification. Table 4 illustrates that our proposed model is faster in both training and testing steps in comparision to stacked LSTM model. It is slower than both ELM and TELM methods due to its complex architecture which enables it to outperform them in accuracy. In training, we ran the algorithms 30 times to select the 10 best runs and vote between them. It costs about 70.29 s to train these models which is about 10 times faster than the learning time of the stacked LSTM model which needs an additional 669 s to train. The detection time per sample of our proposed method is less than the stacked LSTM, too. It costs 1.2 ms for a sample to pass the proposed model which is faster than the 2.7 ms of the stacked LSTM.

The modified TELM model is worked similar to the CNN model since both considered local dependencies in the first stage and pooled them to complete the model. But they are different in calculations since the convolution between input and filter is applied in the CNN model but we use a non-linear activation function instead. Also the second layer of the proposed model works as a pooling layer by aggregating the local features to make the global features. In modified TELM, we use this layer to classify the samples. Then, we compared the performance of the proposed method with these three CNN models. As illustrated in Table 3, the CNN model outperformed the three models when evaluated using the VXHeaven and Kaggle datasets. Also, it yields better AUC in almost all the datasets used in the evaluation; thus, implying that this model obtains better results for all thresholds. The proposed method performs better than CNN in accuracy and MCC in all cases, with the exception when VXHeaven and Kag-

Table 3
Comparison of the proposed method to standard stacked LSTM, CNN, ELM, and TELM. The number in the parenthesis is the standard deviation.

Dataset	Method	ACC	AUC	MCC
Ransomware (System Call)	Proposed Method	**98.96%(0.7)**	0.989 (0.016)	**0.974 (0.018)**
	Stacked LSTM	93.75%(1.59)	0.984 (0.010)	0.911 (0.016)
	One-Layer CNN	98.54%(1.41)	**0.992 (0.004)**	0.968 (0.031)
	Two-Layer CNN	97.27%(0.89)	0.990 (0.020)	0.937 (0.021)
	Three-Layer CNN	96.09%(1.47)	0.989 (0.002)	0.928 (0.033)
	ELM	90.11%(3.68)	0.827 (0.107)	0.779 (0.094)
	TELM	76.39% (12.16)	0.825 (0.106)	0.587 (0.241)
IoT (Opcode)	Proposed Method	**99.65% (0.6)**	**0.995 (0.014)**	**0.992 (0.014)**
	Stacked LSTM	93.69% (1.87)	0.976 (0.013)	0.830 (0.021)
	One-Layer CNN	98.20%(0.48)	0.993 (0.002)	0.965 (0.158)
	Two-Layer CNN	98.20%(1.27)	**0.995 (0.003)**	0.964 (0.025)
	Three-Layer CNN	97.74%(1.86)	0.992 (0.01)	0.955 (0.037)
	ELM	90.54% (0.64)	0.895 (0.078)	0.819 (0.0002)
	TELM	84.24% (9.55)	0.725 (0.191)	0.701 (0.167)
VXHeaven (Opcode)	Proposed Method	82.40% (0.72)	0.872 (0.019)	0.658 (0.009)
	Stacked LSTM	76.81% (1.63)	0.869 (0.012)	0.590 (0.019)
	One-Layyer CNN	82.52%(0.97)	0.896 (0.004)	0.682 (0.010)
	Two-Layer CNN	87.65%(1.41)	0.935 (0.004)	0.754 (0.027)
	Three-Layer CNN	**88.81%(1.1)**	**0.940 (0.010)**	**0.776 (0.013)**
	ELM	75.06% (1.32)	0.782 (0.028)	0.512 (0.037)
	TELM	69.93% (1.98)	0.776 (0.014)	0.398 (0.043)
Kaggle (Opcode)	Proposed Method	97.29% (0.48)	0.970 (0)	0.941 (0.004)
	Stacked LSTM	98.20% (1.63)	0.866 (0.014)	0.736 (0.029)
	One-Layer CNN	97.57%(1.02)	0.976 (0.012)	0.939 (0.011)
	Two-Layer CNN	98.28%(1.39)	**0.996 (0.004)**	**0.967 (0.025)**
	Three-Layer CNN	**98.35%(1.33)**	0.993 (0.007)	**0.967 (0.026)**
	ELM	85.34% (1.48)	0.883 (0.019)	0.708 (0.003)
	TELM	54.27% (1.81)	0.775 (0.007)	0.227 (0.007)
IoT+Ransomware	Proposed Method	**99.034% (0.49)**	0.983 (0.006)	**0.983 (0.009)**
	Stacked LSTM	95.80% (1.48)	0.974 (0.007)	0.914 (0.029)
	One-Layer CNN	98.79%(0.67)	0.992 (0.004)	0.959 (0.193)
	Two-Layer CNN	97.72%(0.764)	0.994 (0.001)	0.953 (0.016)
	Three-Layer CNN	96.95%(1.21)	**0.995 (0.003)**	0.935 (0.021)
	ELM	90.83% (2.124)	0.897 (0.404)	0.798 (0.043)
	TELM	82.93 (1.96)	0.775 (0.078)	0.634 (0.027)

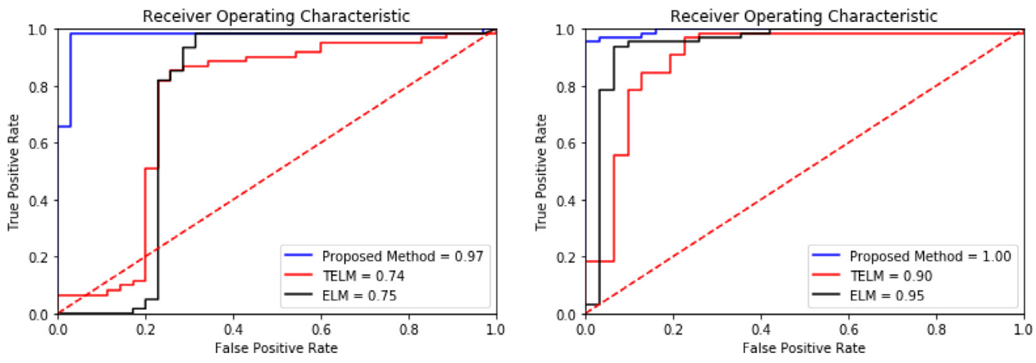

Fig. 5. ROC curve of proposed method, ELM, and TELM for two runs of ransomware dataset.

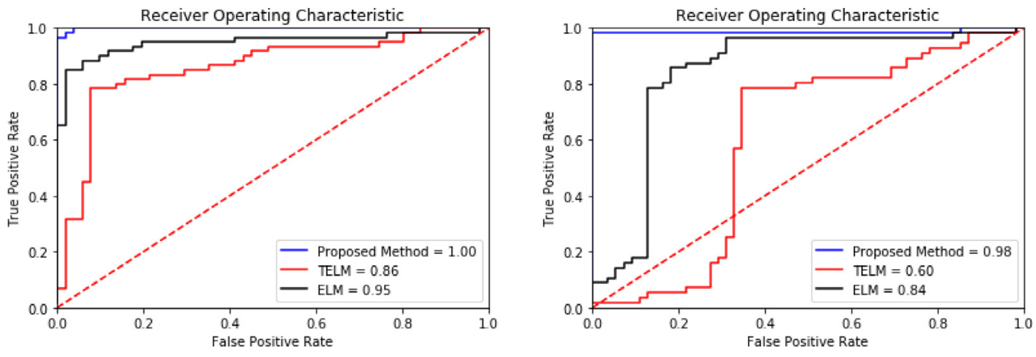

Fig. 6. ROC curve of proposed method, ELM, and TELM for two runs of IoT dataset.

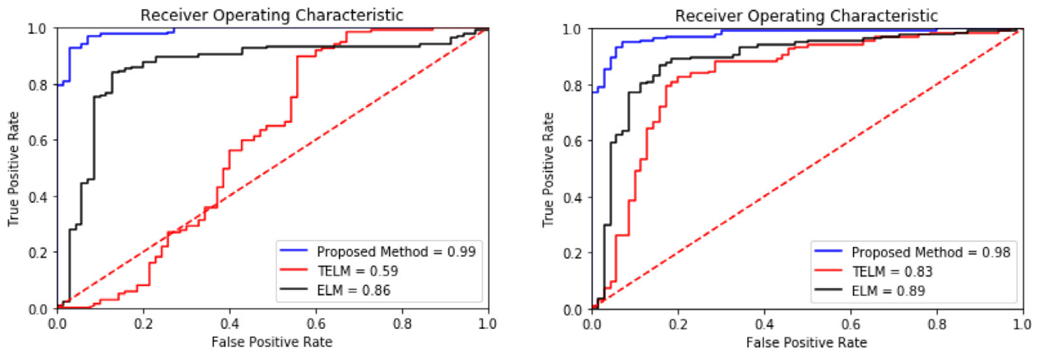

Fig. 7. ROC curve of proposed method, ELM, and TELM for two runs of IoT+Ransomware dataset.

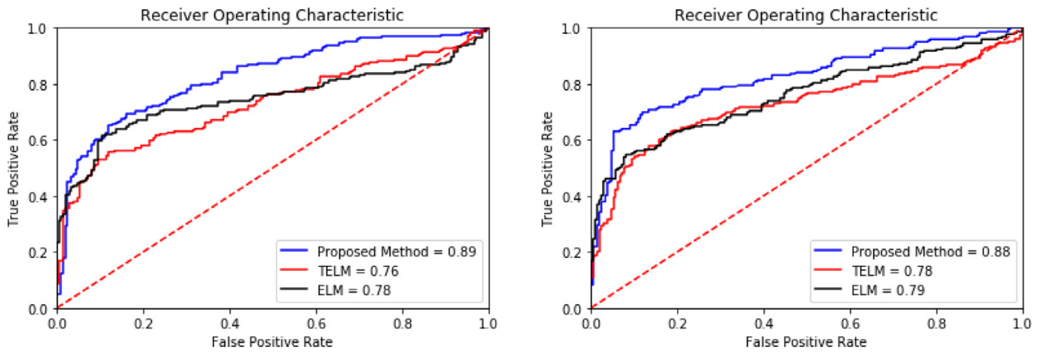

Fig. 8. ROC curve of proposed method, ELM, and TELM for two runs of VXHeaven dataset.

Table 4
Comparison of training time on IoT+Ransomware dataset and detection time per sample of proposed method, stacked LSTM, one-layer CNN, TELM and ELM.

Method	Training Time (Second)	Detection Time per sample (Second)
Proposed Method	2.343	0.0012
Stacked LSTM	739.672	0.0027
Two-Layer CNN	418.36	0.0022
TELM	1.352	0.00093
ELM	0.647	0.00037

gle datasets are used (these two datasets have more samples than other datasets). Hence, one can conclude that as the CNN model goes deeper, an increasing number of model parameters will also require the model to have more training samples to facilitate the tuning of the parameters. Therefore, as the number of layers increases, the performance of small datasets (IoT, Ransomware, and IoT+Ransomware) decreases while the performance of the bigger ones (Kaggle and VXHeaven) increases. However, the proposed method is more robust in case of number of data points and obtain reasonable performance in terms of accuracy, AUC, and MCC with respect to its training and detection time for all datasets, in comparison to other methods.

Moreover, the randomness of the weights and biases between the input layer and the first-hidden layer, without any learning, makes the model very dependent on these weights. Therefore, if we run the ELM, TELM, and modified TELM on a specific dataset using different random initialization, we will likely obtain outcomes with high variances and consequently, the ELM-based models become less robust. To mitigate this limitation, we used an ensemble mechanism in which several random models were built, and among them, the best models were selected for the voting step. In the voting step, the class which was selected with more models was used as our final label.

8. Conclusion and future research

In this paper, we proposed a modified version of a TELM that considers the dependencies of sequence elements so that they can be used for malware detection based on malware samples' raw sequence features (e.g., OpCodes and system calls). We were able to reduce the time for training and sample processing, and yet achieve improved performance in our classification method. When the performance of the proposed approach is compared with those of the standard stacked LSTM, two-layer CNN, ELM, and original TELM on five datasets, the proposed model outperforms original ELM models in terms of accuracy, AUC, and MCC. Specifically, we achieved reduced detection time and improved accuracy rate from 95.80% to 99.03%, AUC from 0.974 to 0.983, and MCC from 0.914 to 0.983 compared to the stacked LSTM. The proposed approach also outperformed the two-layer CNN model in small datasets.

Since this model can handle sequential data, it can be used to applied in other settings such as botnet detection from raw traffic and other sequential data. Moreover, since our model considers local (rather than global) dependencies between input features, additional future work is needed to improve our ELM model and extract the global dependencies of input samples for both malware and botnet detection.

Declaration of Competing Interest

The authors declared that there is no conflict of interest in this study.

References

Alom, M.Z., Sidike, P., Taha, T.M., Asari, V.K., 2016. State preserving extreme learning machine: amonotonically increasing learning approach. Neural Process. Lett. 45, 703–725.

Athiwaratkun, B., Stokes, J.W., 2017. Malware classification with LSTM and GRU language models and a character-level CNN. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 2482–2486. doi:10.1109/ICASSP.2017.7952603.

Azmoodeh, A., Dehghantanha, A., Choo, K.-K.R., 2018. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. IEEE Transactions on Sustainable Computing doi:10.1109/TSUSC.2018.2809665. 1–1, URL http://ieeexplore.ieee.org/document/8302863.

Baldwin, J., Dehghantanha, A., 2018. Leveraging support vector machine for opcode density based detection of crypto-ransomware. In: Cyber Threat Intelligence, pp. 107–136. doi:10.1007/978-3-319-73951-9.

Brunnstein, K., Fischer-Huber, S., Swimmer, M., 1991. Concepts of an expert system for virus detection. Inf. Secur. 391–402.

Cao, J., Hao, J., Lai, X., Vong, C.-M., Luo, M., 2016a. Ensemble extreme learning machine and sparse representation classification. J. Frankl. Inst. 353 (17), 4526–4541. doi:10.1016/j.jfranklin.2016.08.024.

Cao, J., Zhang, K., Luo, M., Yin, C., Lai, X., 2016b. Extreme learning machine and adaptive sparse representation for image classification. Neural Netw. 81, 91–102. doi:10.1016/j.neunet.2016.06.001.

Cao, J., Zhao, Y., Lai, X., Ong, M.E.H., Yin, C., Koh, Z.X., Liu, N., 2015. Landmark recognition with sparse representation classification and extreme learning machine. J. Frankl. Inst. 352 (10), 4528–4545. doi:10.1016/j.jfranklin.2015.07.002.

Chebyshev, V., Sinitsyn, F., Parinov, D., Kupreev, O., Lopatin, E., Liskin, A., 2018. IT Threat Evolution Q3 2018. Statistics. Technical Report. Kaspersky. URL https://securelist.com/it-threat-evolution-q3-2018-statistics/88689.

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., Chen, J., 2018. Detection of malicious code variants based on deep learning. IEEE Transactions on Industrial Informatics doi:10.1109/TII.2018.2822680. 1–1.

Damballa, 2008. 3% to 5% of Enterprise Assets Are Compromised by Bot-driven Targeted Attack Malware. URL https://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html.

Demertzis, K., Iliadis, L., 2016. Bio-Inspired Hybrid Intelligent Method for Detecting Android Malware, pp. 289–304. doi:10.1007/978-3-319-27478-2.

Dovom, E.M., Azmoodeh, A., Dehghantanha, A., Newton, D.E., Parizi, R.M., Karimipour, H., 2019. Fuzzy pattern tree for edge malware detection and categorization in iot. J. Syst. Archit. 97, 1–7. doi:10.1016/j.sysarc.2019.01.017.

Farrokhmanesh, M., Hamzeh, A., 2016. A novel method for malware detection using audio signal processing techniques. In: 2016 Artificial Intelligence and Robotics (IRANOPEN). IEEE, pp. 85–91. doi:10.1109/RIOS.2016.7529495.

Fawcett, T., 2006. An introduction to ROC analysis. Pattern Recognit. Lett. 27 (8), 861–874. doi:10.1016/j.patrec.2005.10.010.

Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C., 2018. Malware visualization for fine-grained classification. IEEE Access 6 (c), 14510–14523. doi:10.1109/ACCESS.2018.2805301. URL https://ieeexplore.ieee.org/document/8290767.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. URL http://www.deeplearningbook.org.

HaddadPajouh, H., Dehghantanha, A., Khayami, R., Choo, K.-K.R., 2018. A deep recurrent neural network based approach for internet of things malware threat hunting. Fut. Gener. Comput. Syst. 85, 88–96. doi:10.1016/j.future.2018.03.007.

Hashemi, H., Hamzeh, A., 2018. Visual malware detection using local malicious pattern. J. Comput. Virol. Hacking Tech. 1–14. doi:10.1007/s11416-018-0314-1.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780. doi:10.1162/neco.1997.9.8.1735.

Homayoun, S., Ahmadzadeh, M., Hashemi, S., Dehghantanha, A., Khayami, R., 2018. BoTShark: A Deep Learning Approach for Botnet Traffic Detection, pp. 137–153. doi:10.1007/978-3-319-73951-9.

Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R., 2017. Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. IEEE Transactions on Emerging Topics in Computing doi:10.1109/TETC.2017.2756908. 1–1.

Huang, G.-B., 2015. What are extreme learning machines? Filling the gap between frank Rosenblatt's dream and John Von Neumann's puzzle. Cognit. Comput. 7 (3), 263–278. doi:10.1007/s12559-015-9333-0.

Huang, G.-B., Wang, D.H., Lan, Y., 2011. Extreme learning machines: a survey. Int. J. Mach. Learn. Cybern. 2 (2), 107–122. doi:10.1007/s13042-011-0019-y.

Huang, G.B., Zhu, Q.Y., Siew, C.K., 2004. Extreme learning machine: a new learning scheme of feedforward neural networks. In: IEEE International Conference on Neural Networks - Conference Proceedings. IEEE, pp. 985–990. doi:10.1109/IJCNN.2004.1380068.

Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., 2006. Extreme learning machine: Theory and applications. Neurocomputing 70 (1-3), 489–501. doi:10.1016/j.neucom.2005.12.126. URL http://linkinghub.elsevier.com/retrieve/pii/S0925231206000385.

Hyvärinen, A., Oja, E., 2000. Independent component analysis: algorithms and applications. Neural Netw. 13 (4–5), 411–430. doi:10.1016/S0893-6080(00)00026-5.

Kabanga, E.K., Kim, C.H., 2018. Malware images classification using convolutional neural network. J. Comput. Commun. 6 (1), 153–158. doi:10.4236/jcc.2018.61016.

Kang, B., Yerima, S.Y., Mclaughlin, K., Sezer, S., 2016. N-opcode analysis for android malware classification and categorization. In: 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). IEEE, pp. 1–7. doi:10.1109/CyberSecPODS.2016.7502343.

Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D., 2018. MalDozer: automatic framework for android malware detection using deep learning. Digital Invest. 24, S48–S59. doi:10.1016/j.diin.2018.01.007.

Karimipour, H., Dehghantanha, A., Parizi, R.M., Choo, K.R., Leung, H., 2019. A deep and scalable unsupervised machine learning system for cyber-attack detection in large-scale smart grids. IEEE Access 7, 80778–80788. doi:10.1109/ACCESS.2019.2920326.

Kephart, J.O., Arnold, W.C., 1994. Automatic Extraction of Computer Virus Signatures.

Kozik, R., 2018. Distributing extreme learning machines with Apache Spark for NetFlow-based malware activity detection. Pattern Recognit. Lett. 101, 14–20. doi:10.1016/j.patrec.2017.11.004.

Le, Q., Boydell, O., Mac, B., Scanlon, M., 2018. Deep learning at the shallow end : malware classification for non-domain experts. In: Digital Investigation. Elsevier, pp. S118–S126.

Li, M., Xiao, P., Zhang, J., 2018. Text Classification Based on Ensemble Extreme Learning Machine arXiv:1805.06525.

Liu, N., Wang, H., 2010. Ensemble based extreme learning machine. IEEE Signal Process. Lett. 17 (8), 754–757. doi:10.1109/LSP.2010.2053356.

Juan Lu, H., lin An, C., ping Ma, X., Zheng, E.-H., bing Yang, X., 2014. Disagreement Measure Based Ensemble of Extreme Learning Machine for Gene Expression Data Classification.

McAfee, 2018. McAfee Labs Thread Report December 2018. Technical Report.

McLaughlin, N., Doupé, A., Joon Ahn, G., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., 2017. Deep android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17. ACM Press, New York, New York, USA, pp. 301–308. doi:10.1145/3029806.3029823.

Meng, X., Shan, Z., Liu, F., Zhao, B., Han, J., Wang, H., Wang, J., 2017. MCSMGS: malware classification model based on deep learning. In: 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE, pp. 272–275. doi:10.1109/CyberC.2017.21.

Microsoft, 2015. Microsoft Malware Classification Challenge. URL https://www.kaggle.com/c/malware-classification.

Mohammadi, S., Mirvaziri, H., Ghazizadeh-Ahsaee, M., Karimipour, H., 2019. Cyber intrusion detection by combined feature selection algorithm. J. Inf. Secur. Appl. 44, 80–88. doi:10.1016/j.jisa.2018.11.007.

Nauman, M., Tanveer, T.A., Khan, S., Syed, T.A., 2018. Deep neural architectures for large scale android malware analysis. Cluster Comput. 21 (1), 569–588. doi:10.1007/s10586-017-0944-y.

Ni, S., Qian, Q., Zhang, R., 2018. Malware identification using visualization images and deep learning. Comput. Secur. 871–885. doi:10.1016/j.cose.2018.04.005.

Preda, M.D., Christodorescu, M., Jha, S., Debray, S., 2008. A semantics-based approach to malware detection. ACM Trans. Program. Lang. Syst. 30 (5), 25:1–25:54. doi:10.1145/1387673.1387674.

Qu, B.Y., Lang, B.F., Liang, J.J., Qin, A.K., Crisalle, O.D., 2015. Two-hidden-layer extreme learning machine for regression and classification. Neurocomputing doi:10.1016/j.neucom.2015.11.009.

Rao, K., Yip, P., 2001. The Transform and Data Compression Handbook. CRC Press.

Santos, I., Brezo, F., Nieves, J., Penya, Y.K., Sanz, B., Laorden, C., Bringas, P.G., 2010. Idea: opcode-sequence-based malware detection. In: Massacci, F., Wallach, D., Zannone, N. (Eds.), Engineering Secure Software and Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 35–43.

Santos, I., Penya, Y.K., Devesa, J., Bringas, P.G., 2009. N-grams-based file signatures for malware detection. ICEIS.

Schliebs, S., Kasabov, N., 2013. Evolving spiking neural network-a survey. Evol. Syst. 4 (2), 87–98. doi:10.1007/s12530-013-9074-9.

Schmidhuber, J., 2015. Deep learning in neural networks: an overwiew. Neural Netw. 61, 85–117.

Shalaginov, A., Banin, S., Dehghantanha, A., Franke, K., 2018. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial, pp. 7–45. doi:10.1007/978-3-319-73951-9.

Smith, L.I., 2002. A tutorial on Principal Components Analysis. Technical Report. University of Otago, Otago, New Zeland.

Sun, Y., Xie, Y., Qiu, Z., Pan, Y., Weng, J., Guo, S., 2017. Detecting android malware based on extreme learning machine. In: 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech). IEEE, pp. 47–53. doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2017.24.

Symantec, 2018. Internet Security Threat Report 23 Volume. Technical Report. Symantec.

Tesauro, G.J., Kephart, J.O., Sorkin, G.B., 1996. Neural networks for computer virus recognition. IEEE Expert 11 (4), 5–6. doi:10.1109/64.511768.

Vinayakumar, R., Soman, K., Poornachandran, P., Sachin Kumar, S., 2018. Detecting android malware using long short-term memory (LSTM). J. Intell. Fuzzy Syst. 34 (3), 1277–1288. doi:10.3233/JIFS-169424.

VXHeaven Virus Collection, URL http://83.133.184.251/virensimulation.org.

Wang, Y., Cao, F., Yuan, Y., 2011. A study on effectiveness of extreme learning machine. Neurocomputing 74 (16), 2483–2490. doi:10.1016/j.neucom.2010.11.030.

Advances in Extreme Learning Machine: Theory and Applications Biological Inspired Systems. Computational and Ambient Intelligence.

Wognsen, E.R., Karlsen, H.S., Olesen, M.C., Hansen, R.R., 2014. Formalisation and analysis of Dalvik bytecode. Sci. Comput. Program. 92, 25–55. doi:10.1016/j.scico.2013.11.037.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K., 2017. Android malware detection based on system call sequences and LSTM. Multimed. Tools Appl. 1–21. doi:10.1007/s11042-017-5104-0.

Xu, K., Li, Y., Deng, R.H., Chen, K., 2018. DeepRefiner: multi-layer android malware detection system applying deep neural networks. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 473–487. doi:10.1109/EuroSP.2018.00040.

Xu, L., Zhang, D., Alvarez, M.A., Morales, J.A., Ma, X., Cavazos, J., 2016. Dynamic android malware classification using graph-based representations. In: 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, pp. 220–231. doi:10.1109/CSCloud.2016.27.

Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., Sakuma, J., 2018. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy - CODASPY '18. ACM Press, New York, New York, USA, pp. 127–134. doi:10.1145/3176258.3176335.

Yan, J., Qi, Y., Rao, Q., 2018. LSTM-based hierarchical denoising network for android malware detection. Secur. Commun. Netw. 2018, 1–18. doi:10.1155/2018/5249190.

Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S., 2017. A survey on malware detection using data mining techniques. ACM Comput. Surv. 50 (3), 41:1–41:40. doi:10.1145/3073559.

Ye, Y., Li, T., Zhu, S., Zhuang, W., Tas, E., Gupta, U., Abdulhayoglu, M., 2011. Combining file content and file relations for cloud based malware detection. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, pp. 222–230. doi:10.1145/2020408.2020448.

Yousefi-Azar, M., Hamey, L., Varadharajan, V., Chen, S., 2018a. Learning Latent Byte-Level Feature Representation for Malware Detection, 3. Springer International Publishing, pp. 568–578. doi:10.1007/978-3-030-04212-7.

Yousefi-Azar, M., Hamey, L.G.C., Varadharajan, V., Chen, S., 2018b. Malytics: a malware detection scheme. IEEE Access 6, 49418–49431. doi:10.1109/ACCESS.2018.2864871.

Yuan, Z., Lu, Y., Xue, Y., 2016. Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci. Technol. 21 (1), 114–123. doi:10.1109/TST.2016.7399288.

Zhang, J., Qin, Z., Yin, H., Ou, L., Xiao, S., Hu, Y., 2016. Malware variant detection using opcode image recognition with small training sets. In: 2016 25th International Conference on Computer Communication and Networks (ICCCN). IEEE, pp. 1–9. doi:10.1109/ICCCN.2016.7568542.

Zhang, W., Ren, H., Jiang, Q., Zhang, K., 2015. Exploring Feature Extraction and ELM in Malware Detection for Android Devices, pp. 489–498. doi:10.1007/978-3-319-25393-0.

Amir Namavar Jahromi is with the Machine Learning Lab, Department of Computer Science, Engineering and Information Technology, Shiraz University, Iran.

Sattar Hashemi is with the Machine Learning Lab, Department of Computer Science, Engineering and Information Technology, Shiraz University, Iran.

Ali Dehghantanha is the director of Security of Advanced Systems (SoAS) lab in the School of Computer Science, University of Guelph (UofG), Ontario, Canada. He has served for more than a decade in a variety of industrial and academic positions with leading players in Cyber-Security and Artificial Intelligence. Prior to joining UofG, he has served as a Sr. Lecturer in the University of Sheffield, UK and as an EU Marie-Curie International Incoming Fellow at the University of Salford, UK. His main research interests are malware analysis and digital forensics, IoT security and application of AI in the Cyber Security.

Kim-Kwang Raymond Choo holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA). He is the recipient of the 2019 IEEE TCSC Award for Excellence in Scalable Computing (Middle Career Researcher), 2018 UTSA College of Business Endowed Research Award for Tenured Faculty, IEEE Access Outstanding Associate Editor of 2018, British Computer Society's 2019 Wilkes Award Runner-up, 2019 EURASIP JWCN Best Paper Award, Korea Information Processing Society's JIPS Survey Paper Award (Gold) 2019, IEEE Blockchain 2019 Outstanding Paper Award, IEEE TrustCom 2018 Best Paper Award, ESORICS 2015 Best Research Paper Award, 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. He is also a Fellow of the Australian Computer Society, an IEEE Senior Member, and Co-Chair of IEEE MTCT's Digital Rights Management for Multimedia Interest Group.

Hadis Karimpour is currently an assistant professor in the School of Computer Science, University of Guelph (UofG), Ontario, Canada. Her research interests include AI for security analysis, smart grid modeling, analysis and security, and intelligent control system modeling and analysis.

David Ellis Newton is with the School of Computer Science, University of Salford, UK.

Reza M. Parizi is the Director of the Decentralized Science Lab (dSL) in the College of Computing and Software Engineering at Kennesaw State University, GA, USA. He is a consummate technologist and software security researcher with an entrepreneurial spirit. He is a Senior IEEE member and a member of the IEEE Blockchain Community and ACM. Prior to joining KSU, he was an associate professor at New York Institute of Technology. He received a Ph.D. in Software Engineering in 2012. His research interests are R&D in blockchain, smart contracts, federated learning, and emerging issues the practice of secure software-run world applications.