Malware Detection Using Deep Recurrent Neural Networks with no Random Initialization

Amir Namavar Jahromi and Sattar Hashemi
a.namavar@aut.ac.ir s_hashemi@shiraz.ac.ir
Shiraz University, Computer Science & Engineering and Information Technology Department, Shiraz, Iran

Summary
Malware detection is an increasingly important operational focus in cyber security, particularly given the fast pace of such threats (e.g., new malware variants introduced every day). There has been great interest in exploring the use of machine learning techniques in automating and enhancing the effectiveness of malware detection and analysis. In this paper, we present a *deep recurrent neural network* solution as a *stacked Long Short-Term Memory (LSTM*) with a pre-training as a regularization method to avoid random network initialization. In our proposal, we use global and short dependencies of the inputs. With pre-training, we avoid random initialization and are able to improve the accuracy and robustness of malware threat hunting. The proposed method speeds up the convergence (in comparison to stacked LSTM) by reducing the length of malware OpCode or bytecode sequences. Hence, the complexity of our final method is reduced. This leads to better accuracy, higher Mattews Correlation Coefficients (MCC), and Area Under the Curve (AUC) in comparison to a standard LSTM with similar detection time. Our proposed method can be applied in real-time malware threat hunting, particularly for safety critical systems such as eHealth or Internet of Military of Things where poor convergence of the model could lead to catastrophic consequences. We evaluate the effectiveness of our proposed method on Windows, Ransomware, Internet of Things (IoT), and Android malware datasets using both static and dynamic analysis. For the IoT malware detection, we also present a comparative summary of the performance on an IoT-specific dataset of our proposed method and the standard stacked LSTM method. More specifically, of our proposed method achieves an accuracy of 99.1% in detecting IoT malware samples, with AUC of 0.985, and MCC of 0.95; thus, outperforming standard LSTM based methods in these key metrics.

Keywords:
Deep learning, Stacked LSTM, Unsupervised layer-wise pre-training, Malware, Cyber Threat Hunting.

1. Introduction

Malicious software (a.k.a. malware) is a program designed to disrupt, damage, or gain unauthorized access to a computing system. The threat of malware to our society is evident, for example by the constant increases in the number and types of malware discovered. In 2018, for example, Symantec reported more than 669 million new malware variants, an increase of 80.1% from the previous year [1]. The number of mobile malware has also reportedly increased by 54% [1]. Similar observation is made by other security companies. For example, McAfee reportedly detected 16 million mobile malware in the first quarter of 2018, nearly doubled in comparison to the previous quarter [2]. Also, the number of newly created variants of ransomware increased by 46% (i.e., from 241 families in 2016 to 350 new ransomware families in 2017) [1].

Machine learning techniques have emerged as promising solutions to tackle the challenge of automated malware detection and analysis [3]. Basic metrics for evaluating performance of machine learning algorithms are true positive (TP), true negative (TN), false positive (FP), and false negative (FN) as shown in Equations (1) to (4), respectively.

$$TP = \sum samples \text{ correctly classified as malware} \tag{1}$$

$$FP = \sum samoples \text{ wrongly classified as malware} \tag{2}$$

$$TN = \sum samples \text{ correctly classified as benign} \tag{3}$$

$$FN = \sum samples \text{ wrongly classified as benign} \tag{4}$$

Using above direct metrics, we can define True Positive Rate (TPR), False Positive Rate (FPR), Accuracy (ACC), Mattews Correlation Coefficients (MCC), Receiver Operating Characteristic (ROC) curve, and Area Under the Curve (AUC) to measure performance of machine learning algorithms in malware detection. True positive rate (TPR) which is also known as recall or probability of malware detection (see Equation (5)). The false positive rate (FPR) which is also known as the fallout is the probability of wrongly detecting a benign sample as a malware (see Equation (6). Accuracy indicates the number of samples that were classified correctly over the

Manuscript received August 5, 2023
Manuscript revised August 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.8.21

entire dataset [4] (see Equation (7). MCC is a number between -1 and +1, in which -1 shows that the classifier wrongly classified samples all the time, +1 means the classifier classified malware correctly all the times, and 0 means that the machine learning algorithm does not work better than random prediction (see Equation (8)). ROC is a curve that shows the TPR and FPR of algorithm in several thresholds. 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) [5]. AUC value of 0 means that the algorithm classifies all samples wrongly, 0.5 shows that the algorithm did not work better than random classifier, and 1 shows that it can classify all the samples correctly.

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$FPR = \frac{FP}{TN + FP} \tag{6}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{8}$$

Initial attempts to use machine learning for malware detection were focused on representing malware features (i.e. OpCodes, library calls, etc.) as an image, audio file, or even a digital signal file and then apply machine learning classifiers on new representation of malware samples (see [6] [7] [8] [9] [10]) . Despite reporting a reasonable accuracy of about 94%, the time required to change samples representation (1.3 to 3 seconds for pre-processing in [8, 10, 11, 12] for instances), and run the test have made these techniques impractical [13]. To address the issue of the time efficiency, several proposals have been in the literature that use dynamic features such as system calls with major classifiers such as k-nearest neighbors (KNN), decision tree, random forest, Nave Bayes, Bayesian network, support vector machine (SVM), and logistic regression for malware detection (see [14, 15, 16, 17]). Others attempted to use natural language processing techniques on malware static features such as OpCode and file sections to identify malicious programs (see [18, 19]). While reducing processing time (by not changing malware representation), these methods were not very accurate (below 87%).

The Convolutional Neural Networks (CNN) as a type of deep learning algorithms were suggested to improve malware detection accuracy. CNN offered a promising accuracy of 92%-99% in detecting malware using a range of dynamic and static features (see [11, 12, 20, 21, 22, 23, 24, 25]). Despite being accurate, CNN-based malware detection systems inevitably hinder from the weaknesses of deep neural networks. The main assumption of feedforward deep neural network techniques with fixed number of input neurons such as CNN is independence and identical distribution of input data [26]. Therefore, the accuracy of CNN- based techniques drops significantly when they deal with variable-length sequence of malware OpCodes or samples with interdependent library calls [27, 28]. To rectify this matter, Recurrent Neural Network (RNN) techniques were built to deal with the variable-length sequential data in deep neural networks [29]. RNN is a multi- layer neural network with tied weights where each layer may receive two inputs, 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 [30]. However, when the length of input data is increased, traditional RNN networks with backpropagation would face the vanishing gradient problem [29], which causes an insufficient learning of initial layers of the deep network. This problem makes the utilization of RNN techniques to be limited to the malware samples with relatively short length of features, i.e. short sequence of OpCode or limited number of library calls [31]. To resolve this issue, while maintaining power of deep networks, stacked Long Short-Term Memory (LSTM) networks were introduced [29]. 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 could deal with almost any length of sequential input data and offer a very good accuracy (97% to 99.7%) - see [32, 33, 34, 35, 36, 37, 38]. However, k-means clustering, Hidden Markov Model (HMM), and stacked LSTM algorithms require random initialization that could lead to poor convergence of the model [39] [40]. Solutions such as using evolutionary algorithms for k-means clustering [41, 42] or making a global generative model for HMM [43, 44] or using better weights for initializing the deep network [40] were suggested to avoid poor convergence caused by random initialization of the model. Murthy and Chowdhury [42] proposed a genetic algorithm method to solve the random initialization of k-means clustering. They used vectors of cluster numbers as their chromosomes each contained the cluster of each data point. Also, [41] tried to avoid random initialization of k-means algorithm by considering the centroids as the chromosomes and try to find the best initialization for k-means algorithm using genetic algorithm technique. Integrating genetic algorithm to

LSTM makes the model very slow since thousands of LSTM models should be trained to gain the best initialization that is not practical. In addition, [43] [44] proposed some models like Maximum Mutual Information Estimation (MMIE) to avoid random initialization in HMM model. Since these models work on the objective function of HMM, and since objective functions of HMM and LSTM are totally different, these models don't work on LSTM method. This is especially important in malware threat hunting, as a poorly initialized LSTM network would reduce the convergence speed of the algorithm and cause the model to stick to a poor local optimum which significantly increases the risk of successful malware attack. To better understand the issues that are caused by random initialization, we have measured the accuracy of a randomly initialized standard LSTM with 4 hidden layers (same setup as other experiments) on 9th class of Kaggle dataset. In the third run, the model converged into a poor situation as shown in Figure 1. Comparing the ROC curve of the poor converged LSTM (red line in Figure 1(a)) with a random classifier with AUC of 0.500 (dotted blackline) shows that the performance of the model is not better than a random guess! Moreover, as shown in Figure 1(b), with poor random initialization, the changes of network weights with backpropagation is so slow that the model could not learn from data at all and the accuracy remains flat.

In this research, we propose a pre-training step for stacked LSTM to avoid random initialization and improve ACC, AUC, ROC, and MCC metrics. Also, the proposed method speeds up the convergence (compare to stacked LSTM) by reducing the length of malware OpCode or bytecode sequences. Hence, the complexity of our final method is reduced which leads to a better performance of accuracy, AUC, and MCC in compare with a standard LSTM with similar detection time.

It is worth mentioning that our proposed method is importantly useful for malware threat hunting in safety critical systems such as eHealth or Internet of Military of Things where poor convergence of the model could lead to a significant damage as re-initialization of the system could be too time consuming.

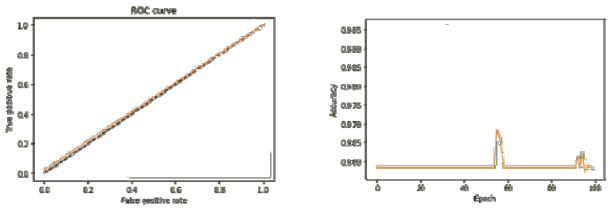

Fig. 1. Example of sticking in local optima for 9th class of Kaggle dataset in (a) ROC curve, and (b) accuracy.

datasets namely, VXHeaven [46] dataset, Kaggle dataset [47], and Windows ransomware dataset [17], as well as an Drebin Android malware dataset [48] and an IoT malware dataset [34]. Our results indicated that our method improves the accuracy between 0.94% and 11.7%, AUC between 0.016 and 0.58, and MCC between 0.089 and 0.215, on various datasets in compare with standard LSTM. Moreover, our methods detection time was 2.7 milliseconds that is much faster than previous comparable methods that reported 1.3 and 12.71 seconds [8, 10, 11]. It is notable that we only tested our model on raw sequences of malware and our results can be extended to sequential data only.

The rest of the paper is organized as follows. In Section 2, the current malware detection methods and their challenges are explained. In Section 3, our malware datasets and experiment setup is explained. Our proposed method is explained in Section 4. Section 5 reports our experiments result followed by discussions in Section 6, which describes our performance in detecting ransomwares families. Finally, section 7 offers conclusions and future works of this paper.

2. Related Work

Many researches have used natural language processing (NLP) techniques for malware detection. Kang et al. [18] organized samples opcodes into feature vectors using N-grams and gain the F-measure of 0.98. Xu et al. [19] extracted n- grams, histogram, and Markov chain from system calls and achieved accuracy of 87.3%. These approaches consider short and local dependencies in malware sequences that is useful for semi-stationary datasets. However, since malware sample sequence are globally related, long dependencies of malware sequence elements may achieve a better [27, 28]. In the recent years, deep learning techniques have been mostly used in many applications domains including malware detection [11, 12, 22, 23, 32, 38, 49]. In IoT malware detection, Azmoodeh et al. [50] used deep eigenspace learning to detect malware of IoT networks. They reported 99.68% of accuracy. Also, Haddadpajouh et al. [34] used deep bidirectional LSTM for this purpose. In their work, they used a bidirectional LSTM instead of standard LSTM to improve the network accuracy and reported 98.18% of accuracy with detection time of 1 second per sample. But the network still suffers from random initialization and the risk of sticking in poor local optimum, which are addressed in this paper.

Two types of deep approaches are used in this area, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN).

CNN based approaches try to learn efficient language model from several short dependencies of input sequences.

These methods are especially suitable for methods with a semi- stationary input such as test or image files. McLaughlin et al. [24] used CNN on malware OpCodes using traditional N-gram extraction and obtained 98% accuracy with 0.0003 seconds detection time per sample using GTX980 GPU. Le et al. [25] used CNN approach on binary codes of samples and reported accuracy of 98.2% with the detection time of 0.02 seconds per sample with GTX1080Ti GPU. Also, [11] used CNN approach for malware detection on their dataset with the accuracy of 98.86% and the detection time of 1.6 to 12.71 seconds on CPU. Among RNN based techniques, Long Short-Term Memory (LSTM) is used more than others in analyzing sequential data due to their robustness on vanishing gradient. Many researches such as [27, 28, 33] have implemented and compared performance of CNN and LSTM methods in malware detection. Their results further indicate better performance of LSTM methods in compare with CNN due to the consideration of longest dependencies extracted from malware data assumed as stationary. However, all existing LSTM based methods require random initialization which significantly limits their application to protecting safety critical systems.

3. Data Preparation and Experiment Setup

We used six datasets from various platforms covering, Windows, Android, and IoT malware. We utilized four Windows malware samples datasets including two datasets extracted from VXHeaven [46]. The first dataset contained 3300 samples that 2200 of them were used for training, and 1100 samples were used for testing of the method. The included samples were taken randomly from the entire dataset and were labeled as malware and benign. These two datasets have been used by fellow researchers previously in [9, 10, 51]. The third Windows dataset was Microsoft Kaggle dataset [47]. Samples of this dataset were labeled as nine families of malware variants without any benign samples. Table 2 shows the number of

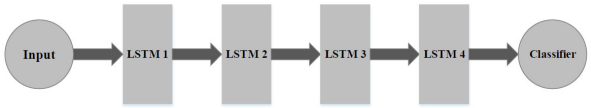

Fig. 2. Final model with four hidden layer for both proposed and stacked method.

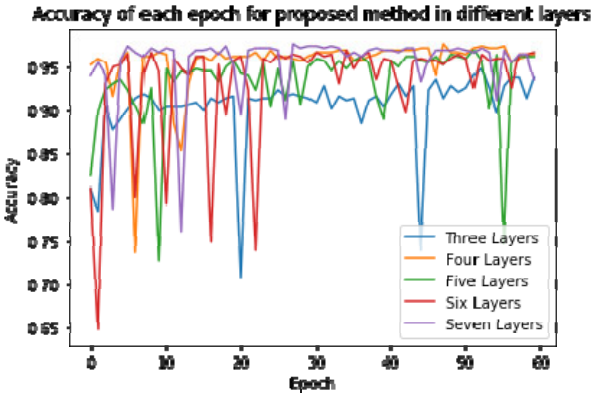

Fig. 3. Comparing proposed networks with 3, 4, 5, 6, 7 hidden layers

samples in each family of this dataset. This dataset contained 10825 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. in [17].

We used Drebin Android malware feature dataset [48] which contained behavioral features such as system calls of 5560 malware and 123453 benign samples. Finally, we used an IoT dataset [34] which contained a total of 552 samples from which 271 samples were labeled as benign and 281 samples were labeled as malware. Table 1 summarizes the basic information and characteristics of all six datasets that we used in this research.

We run all our experiments on a Core i7-4500U desktop with 8 GB of RAM and Windows 10 operating system.

To pass the input data to our model, feature vectors were needed to be crafted in the form of one-hot or embedding vector. To this end, we used an embedding vector for system call and opcode datasets as the neighborhood system calls. However, as the adjacent bytes of bytecode dataset are not related to each other, we chose one-hot vector due to its independency of neighborhood bytes. In our setting, a 64 length for embedding vector for all of our

Table 1. Dataset description

Dataset	Type of Analysis	Type of Data	Number of Classes	Number of Samples
VXHeaven	Static	Bytecode	2	3300
VXHeaven	Static	Opcode	2	3300
Ransomware	Dynamic	System Call	2	1801
Kaggle	Static	Opcode	9	10825
Drebin	Dynamic	System Calls, Permissions, ...	2	129013
IoT	Static	Opcode	2	552

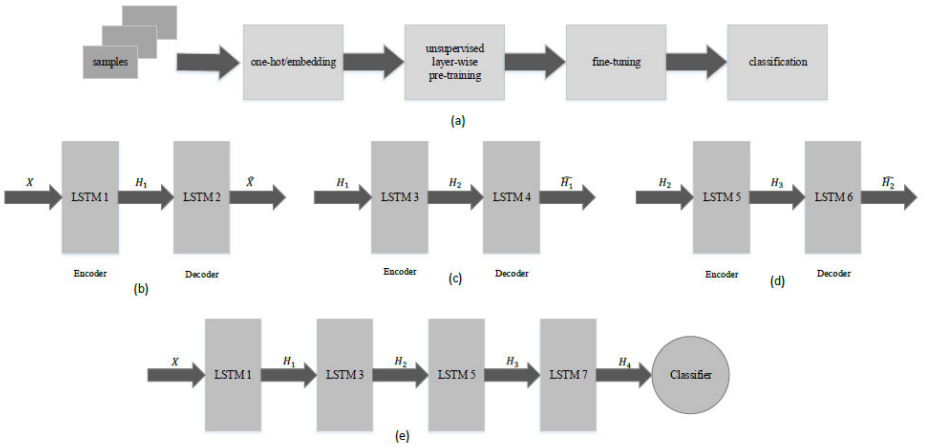

Fig. 4. The proposed method. (a) is steps of proposed method.(b), (c), and (d) are the pre-training phase that reconstruct their input in the output. X is the input data in sequence format, H1 is the first representation learned by network (b), H2 is the second representation learned from H1 in network (c), H3 is the third representation learned from H2 in (d), all of them in sequence format. (e) is the final model with pre-trained layers (weights) from first phase. H4 is learned from H3 with LSTM 6 to map its input sequence to a vector in order to feed the classifier.

dynamic and opcode datasets, and a 256 length one-hot vector for malware bytecode dataset were used. We conducted several experiments to find the best length of vector. The length of one-hot vector is equal to the all possible values of a byte.

Table 2. 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

Our final model consists of four hidden layers as shown in Figure 2. As illustrated in Figure 3, adding or removing hidden layers will reduce the accuracy of the network so a network with 4 hidden layers is selected. As can be seen in Figure 3, a network with four hidden layers (three pre-trained layers) and the network with three hidden layers (two pre-trained layers) are having a comparable accuracy in all but last epoch while the five-layer (four pre-trained layers) network is always performing poorly in compare with a network with four hidden layers. The first three layers are pre-trained LSTMs that feed with sequences and their outputs are sequential with 256, 168, and 128 memory units, respectively. These three layers are responsible for converting the raw input sequence into a new sequential representation. After this non-linear mapping, we needed a layer to make a vector from this new representation to feed the classifier. Thus, we used a LSTM by considering the output of last sequence vector as our final representation. This layer has 64 memory units, so the final representation of our model became a vector with 64 length. Afterwards, we used a dense layer as a classifier. It is important to note that the size of this layer is dependent on the number of classes. For example, a binary classifications can be made with one neuron only

Manuscript received August 5, 2023
Manuscript revised August 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.8.21

while we need nine neurons to classify our Kaggle dataset into nine families.

4. Proposed Method

To resolve the problem of random initialization of deep recurrent networks, we used unsupervised layer-wise pre- training, which has been used previously in methods like stacked autoencoders and deep Boltzmann machine [40]. Our method consists of two steps: 1) pre-training on training data to find initial weights in an unsupervised manner, and 2) fine- tuning the network in a supervised manner to classify samples in malware or benign classes. Figure 4 shows all steps in our proposed.

For sequential data such as bytecodes where the value of neighborhood components of byte string is not important, one-hot representation is used to show each element of the sequence. In this type of representation, each element of sequence is shown by a vector of dictionary size. For example, for bytecode sequences, each element can be between 0 and 255, a 255-sized vector is assigned to each byte of file with all zeros except that equals to the byte value. This vector is assumed as a feature vector for our sequence in a certain time. Because of high length of sequences in the bytecode dataset (1 million for every 1 MB of data), we used the windowing without overlapping method to reduce this size. Our window length is 1024 that means each element of converted sequence is averaging the bytes of 1KB of file. To achieve this, we tried a binary search approach for the window sizes between 10 and 2000. Window sizes between 960 and 1050 had the best results. So, 1024-length window was selected. So, when we use first 100 elements of each sequence, we considered 100 KB of file instead of 100 bytes of file. We use this technique to consider a higher length of sequence without slowing our network.

Moreover, we use an embedding vector to represent datasets that contain system call or opcodes of our samples. An embedding layer was used to map the one-hot vector to a low dimensional vector respect to their neighborhood and frequency. This layer is learned on training data and produced a vector of size 64 for each sequence element as its feature vectors with size of 32 to 128 were tested and 64-length vector had the best result.

Our final models input is a 3-dimensional matrix. For example, if we have n sequence samples with length of l and each element has an m-dimensional feature vector (one-hot or embedding), our input sequence would be n-by-l-by-m. To achieve early detection and speed up our method, we used the first 100 elements of sequence for our classification.

In pre-training step, we use two LSTMs for each layer of our deep network, one for encoding input sequence to a latent space called encoder LSTM, and another for reconstructing the input from the output of first LSTM, called decoder. If each element of input sequence is m-dimensional (size of one- hot or embedding vector), the first LSTM encodes this m- dimensional representation to a p-dimensional representation in the latent space. The goal of this encoder-decoder LSTM is to attain this p-dimensional representation that is more abstract than the input features and extracted automatically obtained by a non-linear function. To learn this layer, a mechanism is needed to adjust the weights with respect to some output. To achieve better generalization, unsupervised learning was selected for pre-training. Therefore, the sequence is considered as an output and the second LSTM (decoder one) is used to map the p-dimensional vectors to m-dimensional ones and reconstruct the input sequence. Backpropagation through time (BPTT) technique [52] is used to adjust the weights based on differences between input sequence and reconstructed sequence. This procedure is repeated in different hidden layers of our main network such that output of each encoder layer is considered as an input for the next encoder-decoder LSTM. For instance, if one wants to create a model with 3 hidden layers, then three encoder-decoder LSTM would be needed for pre-training phase for which the input of first encoder is the input data, the input of the second encoder is the output of first encoder, and the input of third encoder is the output of second encoder (see Figure 4 (a), (b), and (c)). Also, Figure 6 shows the pre-training algorithm. Equation (9) [53, 54, 55], shows the encoding phase where h_t is the encoding result, φ_t^{encode} is the encoder function, x_t is the input of the network at time t (t − th element of sequence), and h_{t-1} is the hidden state of encoder LSTM at time t − 1.

$$H_t = \varphi_t^{encode}(x_t, h_{t-1}) \tag{9}$$

Equation (10) [53, 54, 55] shows the decoder phase of unsupervised pre-training step that $\hat{H}_t$ is the predicted output, φ_t^{decode} is the decoder function, H_t is encoded input at time t, and h_{t-1} is the hidden state of decoder LSTM in the previous time.

$$\hat{H}_t = \varphi_t^{decode}(H_t, h_{t-1}) \tag{10}$$

To train our model in an unsupervised manner, reconstruction error between the true output (input of network) and the predicted output is calculated to update

the weights by BPTT algorithm. This objective function is given in Equation (11) [53, 54, 55].

$$\sum_{t=1}^{T}\left\|\hat{H}_t - H_t\right\| + \beta\left\|W\right\|_2 \quad (11)$$

Where $\hat{H}_t$ is the predicted output, H_t is the network input that is our desired output in unsupervised training, and $\left\|W\right\|_2$ is the $l_2 - norm$ of network weights.

Upon completing pre-training step, the main network is built by concatenating pre-trained layers to each other (see Figure 4 (e)). Now, the model can be fine-tuned in a supervised manner data labels. However, there is a snag in our architecture that the last layers output is in a sequential format. To solve this problem, we used an LSTM layer with vector output and classify its output using a dense layer with softmax activation function. Fine-tuning step is used by BPTT algorithm in respect to the differences of true label and predicted label of each sample (see Figure 7). Equation (12) and (13) [56] show the fine-tuning phase where h_t is the hidden state of LSTM at time t, f is an activation function, x_t is the input of network at time t, and o_t is the output of network at time t. U,W, and V, are weights of network as shown in Figure 5.

$$h_t = f(Ux_t, Wh_{t-1}) \quad (12)$$

$$o_t = f(Vh_t) \quad (13)$$

As illustrated in Equation (12), our network has two inputs at each time, one input from the sequence (x_t) and another from the last internal hidden state of LSTM (h_{t-1}).

The pre-training procedure consists of two LSTM layers (see Figure 6). The first layer is responsible for producing new representation from its input data (variable X from previous layer or input data), which is referred as encoder layer (see Equation 9). The second layer is a decoder (see Equation 10) that reconstructs the input (X) from the output of encoder layer. Also the objective function of this network is the reconstruction error that calculated using Equation 11. This procedure is repeated in different hidden layers of our pre- trained hidden network. In this algorithm, n-th layer is the number of internal neurons of each LSTM layer.

All pre-trained layers are concatenated to make the final network. The final network is fine-tuned in a supervised way to detect malware samples. This fine-tuning is training the feedforward network as well (see Figure 7). The first three layers of this network are the pre-trained layers. Afterwards, a vectorising LSTM is added to the network to convert the final sequence into a vector and pass the output to a dense layer for classification. The network is fine-tuned with backpropagation method [57].

Layer1, Layer2, and Layer3 are pre-trained layers from last step and just added to the final network with all their trained weights. Following these layers, a vectorising layer is added. This layer is an LSTM with vector output. After vectorising, a dense layer is added to the network for classification. The number neurons in this layer equals to the number of classes, except for the binary classification that needs just one neuron. To classify the vectors, softmax activation function is used.

5. Results

To evaluate our method, we chose both static and dynamic malware analysis datasets. As mentioned before, each dataset contains sequences of malware and benign samples including bytecodes, opcodes, and system calls. We compare the performance of our model to the standard stacked LSTM using accuracy, AUC, and MCC metrics.

As can be seen in Table 3, the proposed method achieved better accuracy, AUC, and MCC in compare with standard stacked LSTM in all three datasets.

As AUC and MCC are binary metrics, one vs. all technique is used to evaluate these metrics for Kaggle dataset. In this technique, samples of one class are considered as positive and others are considered as negative class and the model is trained and evaluated accordingly. This procedure is repeated for all the classes and the result is the average of all one versus all models results. As shown in Figure 8, the accuracy of proposed method is higher in all cases.

To visually show the comparison between these two methods, accuracy per epoch and ROC curve of proposed method and standard LSTM are shown in Figure 9 and Figure 10, respectively which indicate better performance of our model. To measure the AUC of Kaggle dataset, one vs. all technique was used (see Figure 11).

Table 4 compares the detection time of our model with similar previous works. It can be seen that our method is the detection time is lesser than all previous models with better performance.

5. Discussion

We compared our method with the same standard stacked LSTM to investigate the effect of our layer-wise

pre- processing of the training phase. As illustrated in Table 3, our method outperformed the standard LSTM in all metrics. As seen in Figure 8, our method converges faster

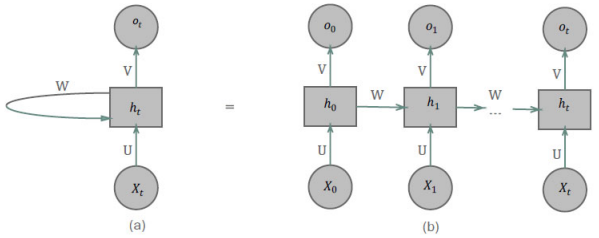

Fig. 5. RNN network. a) A folded RNN. x_t is the sequence input in time t, and O_t is the output of network at time t. The arc arrow shows the recurrently feed of output of network to its input. b) Unfolded representation of an RNN network.

```
procedure PRE-TRAINING
    model = Sequential()
    model.add(LSTM(n^th layer,return_sequences=
True,batch_input_shape=(None,number-of-
features,(n − 1)^t h layer)))
    model.add(LSTM((n −
1)^th layer,return_sequences=True))
model.fit(X,X,nb_epoch,batch_size)
end procedure
```

Fig. 6. Algorithm of pre-training step of proposed method. The first LSTM acts as an encoder and the second one acts as a decoder. The objective function of this algorithm is reconstruction error that calculated using Equation 11.

```
procedure FINE-TUNING
    model = Sequential()
    final-model.add(layer1)
    final-model.add(layer2)
    final-model.add(layer3)
    final-model.add((LSTM(Memory-cells))
    final-model.add(Dense(Number-of-classes, activation=
śoftmax́))
    history=final-model.fit(X, Y, nb_epoch, batch_size)
end procedure
```

Fig. 7. Algorithm of fine-tuning step of proposed method

Table 3. Comparison of proposed method with standard stacked LSTM

Dataset	Method	ACC	AUC	MCC
Ransomware	Proposed Method	**100%**	**1**	**1**
(System Call)	Stacked LSTM	93.75%	0.984	0.911
VXHeaven	Proposed Method	**84.63%**	**0.932**	**0.784**
(Bytecode)	Stacked LSTM	78.97	0.864	0.614
VXHeaven	Proposed Method	**88.51%**	**0.936**	**0.805**
(Opcode)	Stacked LSTM	76.81%	0.869	0.590
Kaggle	Proposed Method	**99.14%**	**0.924**	**0.863**
(Opcode)	Stacked LSTM	98.20%	0.866	0.736
Drebin	Proposed Method	**93.29%**	**0.971**	**0.866**
(Behavioral)	Stacked LSTM	88.13%	0.947	0.763
IoT	Proposed Method	**99.10%**	**0.995**	**0.963**
(Opcode)	Stacked LSTM	93.69%	0.976	0.830

Table 4. Detection time per sample for some methods

Method	Detection Time (Second)	Using GPU
Proposed Method	0.0027	No
IRMD [8]	3	No
LMP [10]	1.3-8.6	No
MCSC [11]	1.61-12.71	No
[22]	0.02	GTX750 Ti
MalDozer [12]	0.003	TITAN X
MalDozer [12]	0.3-0.4	No
CNN-BiLSTM [25]	0.02	GTX1080 Ti
HDN [33]	0.015	GTX980
[35]	1	GPU
DeepRefiner [36]	2.42	GPU

than standard LSTM and it initiates with higher accuracy. Also, the accuracy change of our method is less than the standard LSTM that shows the robustness of our model except in seventh class of Kaggle dataset that is highly imbalanced. A similar trend was seen in Figure 9 for other datasets. This all shows the positive effect of removing random initialization on accuracy and convergence speed of deep recurrent neural networks. As illustrated in Figure 9, the proposed method increased the ACC between 0.94% (for Kaggle dataset) and 11.7% (for VXHeaven dataset) that is related to the higher ACC contribution. Also, it can be seen in Figure 9 (c) and 9(d) that the proposed method reaches its highest accuracy in fewer epochs that shows its faster convergence.

Figure 10 and Figure 11 show the results of ROC and AUC of our method compared to standard LSTM. As seen in these figures, our method has lower false negative rate with higher true positive rates which shows that removing random initialization would lead to better detection of malware samples with lesser false detection. As illustrated in Figure 10 and Table 3, the proposed method increased the AUC between 0.016 (for Ransomware dataset) and 0.068 (for VXHeaven dataset) that is related to the higher AUC contribution.

As shown in Figure 11 (g), none of the models could work well on the seventh class of Kaggle dataset. That is because of the low number of samples in that class. Among 10828 samples of dataset, only 42 samples belong to this class. Standard LSTM predict the samples randomly for this class so the MCC is 0 and the AUC is 0.500. However, the proposed method gained better results even in this highly unbalanced situation by producing the MCC of 0.223 and AUC of 0.550, which is a rather better than random prediction.

As seen in Table 3 and Figure 9 (d) and Figure 10 (d), our proposed method can deal with IoT malware better than the current deep recurrent neural networks. The processing time of our algorithm (in test environment after training) about 3 millisecond (real-time), which is equal to the time for standard LSTM. Hence, we could

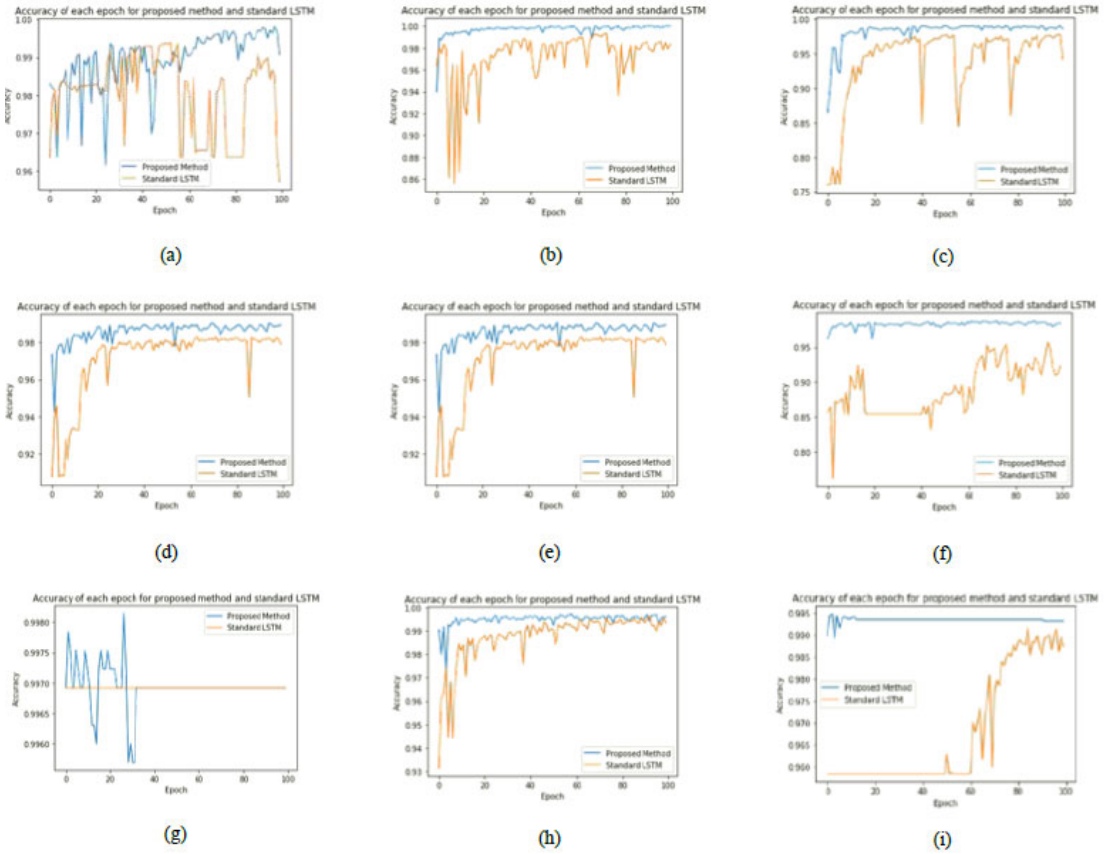

Fig. 8. Comparing accuracy of proposed method and standard LSTM in each epoch for Kaggle dataset classes. (a) first class against all classes, (b) second class against all classes, (c) third class against all classes, (d) fourth class against all classes, (e) fifth class against all classes, (f) sixth class against all classes,
(g) seventh class against all classes, (h) eighth class against all classes, and (i) ninth class against all classes.

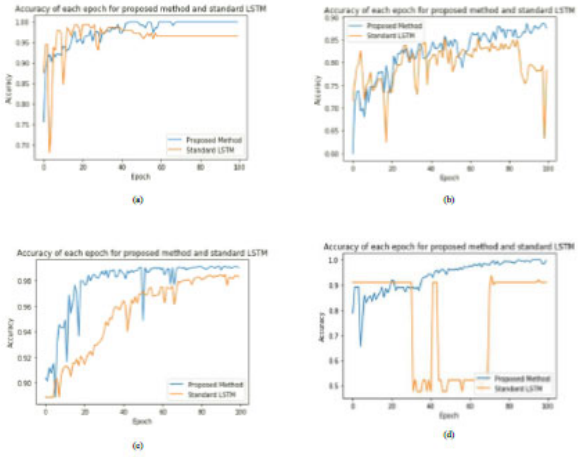

Fig. 9. Comparing accuracy of proposed method and standard LSTM in each epoch. (a) Ransomware dataset, (b) VXHeaven dataset, (c) Kaggle dataset, and (d) IoT dataset.

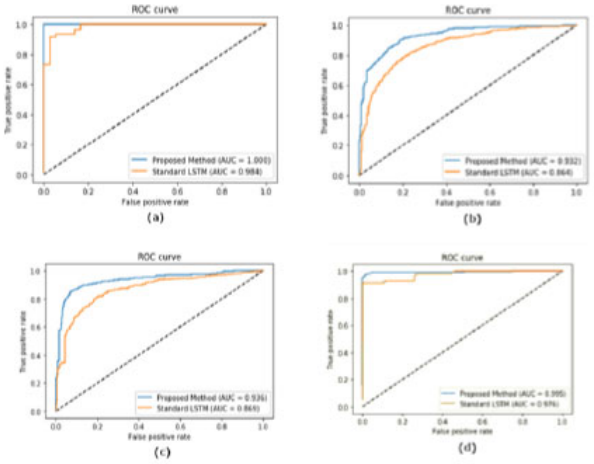

Fig. 10. Comparing AUC and ROC curve of proposed method and stacked LSTM for (a) Ransomware dataset, (b) VXHeaven (bytecode) dataset, (c) VXHeaven (opcode) dataset, and (d) IoT dataset

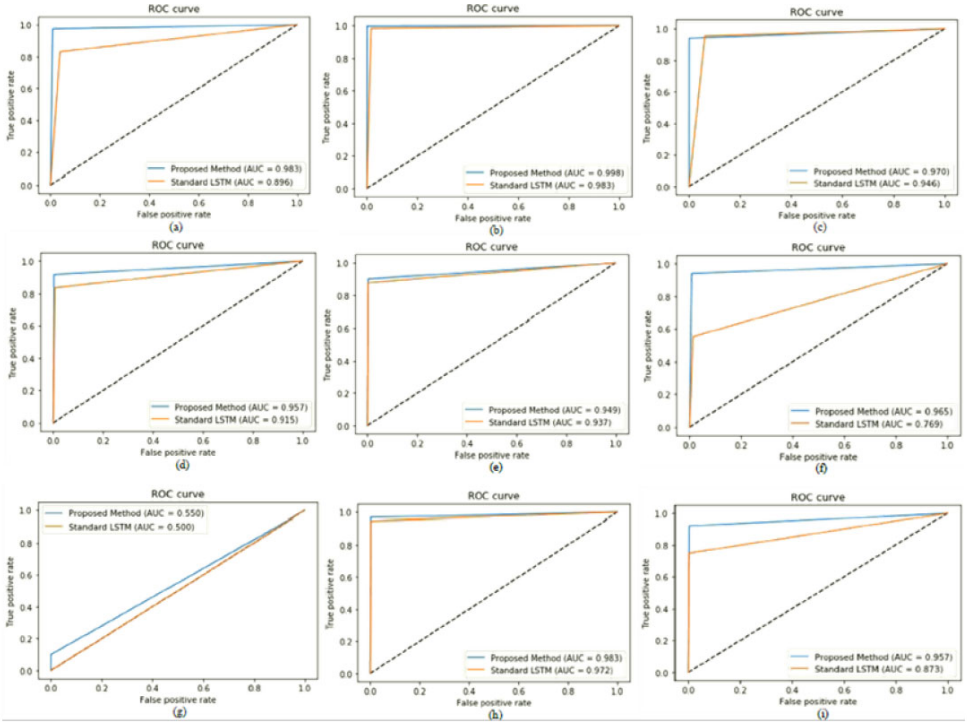

Fig. 11. Comparing AUC and ROC curve of proposed method and standard LSTM in each epoch for Kaggle dataset classes. (a) first class against all classes,
(b) second class against all classes, (c) third class against all classes, (d) fourth class against all classes, (e) fifth class against all classes, (f) sixth class against all classes, (g) seventh class against all classes, (h) eighth class against all classes, and (i) ninth class against all classes.

improve accuracy and all other evaluation metrics without increasing the sample processing time. Attested by this outcome, we believe that our framework can be used in enterprise IoT networks for real-time malware detection with high efficiency.

To show the detection speed of our model, we compared it with several recent peer methods ([8, 10, 11, 12, 22, 25, 33, 36]). As seen in Table 4, our proposed method has the lowest detection time due to its use of raw sequential data, shorter sample data (sequence length), and simple architecture of the network. The speed of our method is similar to MalDozer speed that uses of TitanX GPU while we have not used any GPU in our testing!

5. Conclusion and future work

In this paper, we proposed a stacked LSTM method that not only is considered long dependencies of malware sequence elements but it also avoids random initialization of weights of the network. We used a stacked LSTM with four LSTM layers and a classifier that its first three layers are pre-trained in an unsupervised manner. We not only reduced the time of sample processing but increased the performance of our classification method. We evaluated our method in malware detection space and compared it to the standard stacked LSTM on six diverse datasets. As results indicated, our model outperformed the standard stacked LSTM model in terms of accuracy, AUC, and MCC. Since the detection in the proposed method is real-time, we focused more on IoT application and by fixing the time we could improve the accuracy from 93.69% to 99.10%, AUC from 0.976 to 0.995, and MCC from 0.830 to 0.963.

As we see our work as a framework for deep recurrent learning, we think that it can be used for testing all sequential datasets. Hence, testing this framework on other applications like text-based datasets, time series and bioinformatics is one of our future works. Moreover, using multiple modalities (views) of malware samples may be pursued in the future. Our method can be viewed as a part of a bigger model that combines multiple modalities of malware samples and classify them more efficiently. In addition, because of fast convergence of our method, it can be explored for adversarial malware detection.

References

[1] Symantec, "Internet security threat report 23 volume," Symantec, Tech.
Rep., 2018.
[2] McAfee, "Mcafee mobile threst report," McAfee, Tech. Rep., 2018. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf
[3] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke, "Machine learning aided static malware analysis: A survey and tutorial," 2018, pp. 7–45. [Online]. Available: http://link.springer.com/10.1007/978-3-319-73951-9_2
[4] A. G. Ramirez, C. Lara, L. Betev, D. Bilanovic, U. Kebschull, and f. t. A. Collaboration, "Arhuaco: Deep learning and isolation based security for distributed high-throughput computing," 2018. [Online]. Available: http://arxiv.org/abs/1801.04179
[5] T. Fawcett, "An introduction to roc analysis," Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, jun 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S016786550500303X
[6] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware Visualization for Fine-Grained Classification," IEEE Access, vol. 6, pp. 14 510–14 523, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8290767/
[7] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," International Journal of Information Security, vol. 14, no. 1, pp. 1–14, Feb 2015. [Online]. Available: http://link.springer.com/10.1007/s10207-014-0242-0
[8] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao, and Y. Hu, "Malware variant detection using opcode image recognition with small training sets," in 2016 25th International Conference on Computer Communication and Networks (ICCCN). IEEE, Aug 2016, pp. 1–9. [Online]. Available: http://ieeexplore.ieee.org/document/7568542/
[9] M. Farrokhmanesh and A. Hamzeh, "A novel method for malware detection using audio signal processing techniques," in 2016 Artificial Intelligence and Robotics (IRANOPEN). IEEE, Apr 2016, pp. 85–91. [Online]. Available: http://ieeexplore.ieee.org/document/7529495/
[10] H. Hashemi and A. Hamzeh, "Visual malware detection using local malicious pattern," Journal of Computer Virology and Hacking Techniques, pp. 1–14, Jan 2018. [Online]. Available: http://link.springer.com/10.1007/s11416-018-0314-1
[11] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," Computers & Security, pp. 871–885, Apr 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167404818303481
[12] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," Digital Investigation, vol. 24, pp. S48–S59, Mar 2018. [Online]. Available: http://arxiv.org/abs/1712.08996http://linkinghub.elsevier.com/retrieve/pii/S1742287618300392
[13] J. Baldwin and A. Dehghantanha, "Leveraging support vector machine for opcode density based detection of crypto-ransomware," 2018, pp. 107–136. [Online]. Available: http://link.springer.com/10.1007/ 978-3-319-73951-9_6
[14] W. Mao, Z. Cai, D. Towsley, Q. Feng, and X. Guan, "Security importance assessment for system objects and malware detection," Computers & Security, vol. 68, pp. 47–68, Jul 2017. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167404817300354
[15] N. Nissim, Y. Lapidot, A. Cohen, and Y. Elovici, "Trusted system-calls analysis methodology aimed at detection of compromised virtual machines using sequential mining," Knowledge-Based Systems, vol. 153, pp. 147–175, Aug 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0950705118302041
[16] B. Alsulami, A. Srinivasan, H. Dong, and S. Mancoridis, "Lightweight behavioral malware detection for windows platforms," in 2017 12th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, Oct 2017, pp. 75–81. [Online]. Available: http://ieeexplore.ieee.org/document/8323959/
[17] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, and R. Khayami, "Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence," IEEE Transactions on Emerging Topics in Computing, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8051108/
[18] B. Kang, S. Y. Yerima, K. Mclaughlin, and S. Sezer, "N-opcode analysis for android malware classification and categorization," in 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). IEEE, Jun 2016, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/7502343/
[19] L. Xu, D. Zhang, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic android malware classification using graph-based representations," in 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, Jun 2016, pp. 220–231. [Online]. Available: http://ieeexplore.ieee.org/document/7545923/
[20] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "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 - CODASPY18. New York, New York, USA: ACM Press, 2018, pp. 127–134. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3176258.3176335
[21] E. K. Kabanga and C. H. Kim, "Malware images classification using convolutional neural network," Journal of Computer and Communications, vol. 06, no. 01, pp. 153–158, 2018. [Online]. Available: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jcc.2018.61016
[22] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," IEEE Transactions on Industrial Informatics, pp. 1–1, 2018. [Online].

Available: http://ieeexplore.ieee.org/document/8330042/
[23] X. Meng, Z. Shan, F. Liu, B. Zhao, J. Han, H. Wang, and J. Wang, "Mcsmgs: Malware classification model based on deep learning," in 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE, Oct 2017, pp. 272–275. [Online]. Available: http://ieeexplore.ieee.org/document/ 8250369/
[24] N. McLaughlin, A. Doupe, G. Joon Ahn, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, and Z. Zhao, "Deep android malware detection," in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY17. ACM Press, 2017, pp. 301–308. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3029806.3029823
[25] Q. Le, O. Boydell, B. Mac, and M. Scanlon, "Deep learning at the shallow end : Malware classification for non-domain experts," in Digital Investigation. Elsevier, 2018, pp. S118–S126.
[26] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[27] M. Nauman, T. A. Tanveer, S. Khan, and T. A. Syed, "Deep neural architectures for large scale android malware analysis," Cluster Computing, vol. 21, no. 1, pp. 569–588, Mar 2018. [Online]. Available: http://link.springer.com/10.1007/s10586-017-0944-y
[28] B. Athiwaratkun and J. W. Stokes, "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, mar 2017, pp. 2482–2486. [Online]. Available: http://ieeexplore.ieee.org/document/7952603/
[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov 1997. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735
[30] J. Schmidhuber, "Deep learning in neural networks: An overwiew," Neural Networks, vol. 61, pp. 85–117, 2015.
[31] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, Khayami, K.-K. R. Choo, and D. E. Newton, "Drthis: Deep ran- somware threat hunting and intelligence system at the fog layer," Journal of Future Generation Computer Systems, pp. 94–104, 2019.
[32] S. Homayoun, M. Ahmadzadeh, S. Hashemi, A. Dehghantanha, and R. Khayami, "Botshark: A deep learning approach for botnet traffic detection," 2018, pp. 137–153. [Online]. Available: http://link.springer.com/10.1007/978-3-319-73951-9_7
[33] J. Yan, Y. Qi, and Q. Rao, "Lstm-based hierarchical denoising network for android malware detection," Security and Communication Networks, vol. 2018, pp. 1–18, 2018. [Online]. Available: https://www.hindawi.com/journals/scn/2018/5249190/
[34] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for internet of things malware threat hunting," Future Generation Computer Systems, vol. 85, pp. 88–96, Aug 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X1732486X
[35] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and lstm," Multimedia Tools and Applications, pp. 1–21, Sep 2017. [Online]. Available: http://link.springer.com/10.1007/s11042-017-5104-0
[36] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, Apr 2018, pp. 473–487. [Online]. Available: https://ieeexplore.ieee.org/document/8406618/
[37] R. Vinayakumar, K. Soman, P. Poornachandran, and S. Sachin Kumar, "Detecting android malware using long short-term memory (lstm)," Journal of Intelligent & Fuzzy Systems, vol. 34, no. 3, pp. 1277–1288, mar 2018. [Online]. Available: http://www.medra.org/servlet/aliasResolver?alias=iospress{\&}doi=10.3233/JIFS-169424
[38] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," Tsinghua Science and Technology, vol. 21, no. 1, pp. 114–123, Feb 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7399288/
[39] R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis. New York, New York, USA: John Wiley and Sons, 1973.
[40] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and Bengio, "Why does unsupervised pre-training help deep learning," Journal of Machine Learning Research, vol. 11, pp. 625–660, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1756025
[41] S. Bandyopadhyay and U. Maulik, "An evolutionary technique based on k-means algorithm for optimal clustering in r," Information Sciences, vol. 2002, no. 146, pp. 221–237.
[42] C. Murthy and N. Chowdhury, "In search of optimal clusters using genetic algorithms," Pattern Recognition Letters, vol. 17, no. 8, pp. 825–832, 1996.
[43] D. Povey and P. Woodland, "Minimum phone error and i-smoothing for improved discriminative training," in IEEE International Conference on Acoustics Speech and Signal Processing. IEEE, may 2002, pp. I–105–I–108. [Online]. Available: http://ieeexplore.ieee.org/document/5743665/
[44] L. Bahl, P. Brown, P. de Souza, and R. Mercer, "Maximum mutual information estimation of hidden markov model parameters for speech recognition," in ICASSP86. IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 11. Institute of Electrical and Electronics Engineers, pp. 49–52. [Online]. Available: http://ieeexplore.ieee.org/document/1169179/
[45] M. Henaff, A. Szlam, and Y. Lecun, "Recurrent orthogonal networks and long-memory tasks," in 33rd International Conference on Machine Learning, 2016, pp. 2034–2042.
[46] "Vxheaven virus collection." [Online]. Available: http://83.133.184.251/ virensimulation.org/
[47] Microsoft, "Microsoft malware classification challenge," 2015. [Online]. Available: https://www.kaggle.com/c/malware-classification
[48] D. Arp, M. Spreitzenbarth, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in Network and Disttributed System Security Symposium (NDSS), 2014, pp. 1–15.
[49] S. Huda, S. Miah, J. Yearwood, S. Alyahya, H. Al-Dossari, and R. Doss, "A malicious threat detection model for cloud assisted internet of things (cot) based industrial control system (ics) networks using deep belief network," Journal of Parallel and Distributed Computing, vol. 120, pp. 23–31, Oct 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0743731518302442
[50] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning," IEEE Transactions on Sustainable Computing, pp. 1–1, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8302863/
[51] H. Hashemi, A. Azmoodeh, A. Hamzeh, and S. Hashemi, "Graph embedding as a new approach for unknown malware detection," Journal of Computer Virology and Hacking Techniques, vol. 13, no. 3, pp. 153–166, Aug 2017. [Online]. Available: http://link.springer.com/10.1007/s11416-016-0278-y
[52] P. Werbos, "Backpropagation through time: what it does and how to do it," Proceedings of the IEEE, vol. 78, no. 10, pp. 1550–1560, 1990. [Online]. Available: http://ieeexplore.ieee.org/document/58337/
[53] I. M. Byatas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou, "Patient subtyping via time-aware lstm networks," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 17. ACM Press, 2017, pp. 65–74. [Online]. Available: https://dl.acm.org/citation.cfm?id=3097997
[54] F. Zhao, J. Feng, J. ZHao, W. Yang, and S. Yan, "Robust LSTM-Autoencoders for Face De-Occlusion in the Wild," IEEE Transactions on Image Processing, vol. 27, no. 2, pp. 778–790, 2018.
[55] E. Marchi, F. Vesperini, S. Squartini, and B. Schuller, "Deep Recurrent Neural Network-Based Autoencoders for Acoustic Novelty Detection," Computational Intelligence and Neuroscience, vol. 2017, pp. 1–14, 2017.
[56] W. Bao, J. Yue, and Y. Rao, "A Deep Learning Framework for Financial Time Series Using Stacked Autoencoders and Long-Short Term Mem- ory," PLoS ONE, vol. 6, no. 12, pp. 1–24, 2017.

[57] R. HECHT-NIELSEN, "Theory of the backpropagation neural network," in Neural Networks for Perception. Elsevier, 1992, pp. 65–93. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/B9780127412528500108

Amir Namavar Jahromi is a Ph.D candidate of Artificial intelligence at Shiraz University since 2012. He also has a master's degree in Information Technology form Amirkabir University of Technology (Tehran Polytechnic) of Tehran and a bachelor's degree in Information Technology. He is currently in charge of Machine Learning Laboratory (MLL) in Shiraz University. His research interests are Deep neural networks, Extreme Learning Machine (ELM), and Machine Learning Applications in Computer Security and Computer Networks.

Sattar Hashemi received the PhD degree in computer science from Iran University of Science and Technology in conjunction with Monash University, Australia, in 2008. Following academic appointments at Shiraz University, he is currently an associate professor at Electrical and Computer Engineering School, Shiraz University, Shiraz, Iran. His research interests include machine learning, data mining, social networks, data stream mining, game theory, and adversarial learning.