Journal of Intelligent & Fuzzy Systems 34 (2018) 1277–1288
DOI:10.3233/JIFS-169424
IOS Press

Detecting Android malware using Long Short-term Memory (LSTM)

R. Vinayakumar[a,*], K.P. Soman[a], Prabaharan Poornachandran[b] and S. Sachin Kumar[a]
[a]*Centre for Computational Engineering and Networking (CEN), Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, Amrita University, India*
[b]*Center for Cyber Security and Networks, Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, Amrita University, India*

Abstract. Long Short-term Memory (LSTM) is a sub set of recurrent neural network (RNN) which is specifically used to train to learn long-term temporal dynamics with sequences of arbitrary length. In this paper, long short-term memory (LSTM) architecture is followed for Android malware detection. The data set for evaluation contains real known benign and malware applications from static and dynamic analysis. To achieve acceptable malware detection rates with low computational cost, various LSTM network topologies with several network parameters are used on all extracted features. A stacked LSTM with 32 memory blocks containing one cell each has performed well on detection of all individual behaviors of malicious applications in comparison to other traditional static machine learning classifier. The architecture quantifies experimental results up to 1000 epochs with learning rate 0.1. This is primarily due to the reason that LSTM has the potential to store long-range dependencies across time-steps and to correlate with successive connection sequences information. The experiment achieved the Android malware detection of 0.939 on dynamic analysis and 0.975 on static analysis on well-known datasets.

Keywords: Android malware detection: static and dynamic analysis, deep learning: recurrent neural network (RNN), Long Short-term Memory (LSTM)

1. Introduction

Recent advancements in mobile technologies have made tremendous growth in mobile devices, particularly when smartphones and tablets are being served as a ‘tiny computer’. Android is the most commonly used mobile platform for smartphones and the current market leader with a market share holding nearly 87.6% [1]. With the ease of use, low-cost, and portability nature to be the characteristics, these have been predominantly used as a primary interface to access internet services. Smartphones with internet services has led to new possibilities in people’s daily life activities through numerous applications (apps). Most popular feature-rich apps that accesses internet via Android are social network apps, gaming apps, information exchange apps, cloud storage apps, financial transactions and banking apps etc. As the usage of smart phones surge past the personal computers (PC’s), the malware writers also followed suit, focusing their attention creating malware for the smartphones. Unlike, the long known security mechanisms of PC’s, such as intrusion detection system (IDS), firewalls, encryption antivirus and other end-point based security measures such mechanisms on Android smartphones are just beginning to be introduced. This adds to the ease of spreading malware on the smartphones than the PC’s. TrendLabs has announced top ten installed malicious Android apps

*Corresponding author. R. Vinayakumar, Centre for Computational Engineering and Networking (CEN), Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, Amrita University, India. E-mail: vinayakumarr77@gmail.com.

containing around 71,250 installations in 3Q 2012 security roundup [3]. As reported by Cisco, Android has encountered 99% new malware apps in 2014 [2]. Due to the complex nature of fragmented Android smartphone eco-system where each original equipment manufacturer (OEM) builds and compiles their own firmware, this situation warrants novel security mechanisms to detect the malware and classify them to a specific malware family.

Android uses Google play as an official market store that hosts the apps and there are more than hundred third-party app stores that also host the Android apps. This includes the app marketplaces that are run by the mobile handset manufacturers. While other smartphone app market place like apple IOS (iPhone OS) that enforces stringent control on the apps. The application checks of Google play store are much more lenient, For example, Android market places have mechanisms such as Bouncer [4] to continuously scan for the suspicious application behavior in emulated environment. This has resulted in the reduction of malicious apps. In [4] discussed the detailed functionalities of Android's bouncer. However, bouncer failed to analyze and detect the several vulnerabilities of uploaded apps. Android operating system (OS) has several built-in security mechanisms such as Android permission systems that are meant to control the app permissions. During the app installation, the Android permission systems seeks explicit permission from the users to access any sub-systems. However, most of the users follow the blind approach to provide grant permissions during the installation procedure of apps as the impact is less known to the end-user. Moreover, this process is tedious and doesn't have much option to provide selective permissions. Since the intention of the app is difficult to identify, the damage could be serious including compromise of user data, identify theft and taking over the control of the entire device. Secondly, the set of permissions might be same for both the benign and malicious apps. Hence, Android permission system can be considered as an initial shelter for risk assessment rather than malware detection.

Smartphones security follows the similar techniques and suffers the challenges faced by computers over the years. During the initial years of smartphone era, the most common security solutions were being downsized to fit for smartphones. These downsized PC's based security solutions are inadequate to the smartphones due to excessive resource utilization thereby fails to provide the needed security [5]. There are two fundamental taxonomies of malware detection techniques for smartphones. They are static and dynamic analysis respectively. They are categorized based on the features set of the apps. Static analysis collects set of features from apps by unpacking or disassembling them without the run time execution [17]. By contrast, dynamic analysis examines the run-time execution behavior of apps such as system calls, network connections, memory utilization, power consumption and user interactions, etc. [6]. In order to avoid the limitations of both static and dynamic analysis, researchers and commercial systems have used combination of both the mechanisms that has been termed as hybrid analysis. Hybrid analysis is a two step process where initially static analysis is performed before the dynamic one. With the following advantageous such as less computational cost, low resource utilization, light-weight and less time consuming in nature, the static analysis can be used on devices which are low in memory and processing power. However, dynamic analysis helps to detect fertile malwares notably metamorphic and polymorphic. Metamorphic and polymorphic malwares have the capability to change the code in each and every generation of the same malware family [7]. Hybrid analysis approach is increasingly being used by antivirus providers for the smartphones as it provides higher detection rates [8].

There is a sudden surge in Android malware and this sheer number of new malware instances requires newer approaches as writing signature for each malware is a daunting task. While signature-based or heuristics-based detection are important, usage of machine learning algorithms for the analysis and detection of growing Android malware are increasingly being studied [7, 9, 10]. These machine learning approaches are important to learn the hidden patterns in large-scale data to identify the previously unencountered malicious apps. Machine learning methods rely on collection of feature set. The set of features with a corresponding class is used to train a model to create a separating plane between the benign and malicious apps. This separating plane helps to detect malware and categorize it in to corresponding malware family. However, deep learning is a complex model of machine learning and appeared as an intimidating mechanism with the remarkable results in long-standing artificial intelligence (AI) tasks related to the field of computer vision, natural language processing and speech processing [11]. It's emerging as an active area of research. These novel mechanisms are used in many Cybersecurity tasks such as intrusion detection [12], binary analysis [13], malware detection [14], and traffic identification [15]. This paper uses the recurrent neural network (RNN) technique of deep learning approach particularly long short-term memory (LSTM) [16, 32, 39] for analysis and classification of malicious apps in Android devices. To achieve higher detection rates and to understand the effectiveness of LSTM, various LSTM network topologies have used.

The following sections are organized as follows; Section 2 reviews the related work on static and dynamic Android malware detection and classification. Section 3 describes the concepts of RNN and how they are trained. Section 4 contains the proposed methodology and its experiments. Section 5 displays the detailed evaluation results for various network topologies of LSTM. Future work directions, limitations and conclusion are placed in Section 6.

2. Related work

Recently, applying machine learning and deep learning approaches towards the analysis and categorization of Android malware has become an active area of research. This section studies the related works of recurrent neural network (RNN) and machine learning and neural network approaches typically termed as deep learning in recent days for static and dynamic analysis in Android malware detection and classification.

In the Android apps, a file named AndroidManifest.xml defines all the permissions and the API calls made by the corresponding application [4]. Most work in static analysis relies on AndroidManifest.xml file for features such as permission and API calls. In addition to these two features, other meta-data features such as version name, version number, publisher name etc. has been used [17]. In [18] proposed a malware detection for Android that use one-class support vector machine as machine learning classifier with the features obtained from a set of permissions and control flow graphs (CFG) in static analysis. They studied 2,081 benign apps and 91 malicious apps. The study resulted in high false positives rates of malware detection. In [19] reported ensemble classifier for Android malware detection with extracted features such as permissions and API calls and combination of both permissions and API calls from around 1250 benign apps and 1260 malicious apps. A static analysis framework has been introduced for detection of malicious behaviors by using permissions as features with 6 machine learning classifiers [17]. To see how we can enhance the Android malware detection rate we used the same dataset in this work.

Most work in dynamic analysis of Android malware used system call events at kernel-level as features. As an example, the methods [20, 21] claimed that the system calls are more effective with the experiments that analyzed 230 apps using a prototype. Additionally, the prototype system achieved more success in detecting the malicious patterns of the unknown apps. Recently, [22] surveyed the dynamic malware analysis techniques used in the past years. In [23] presented a novel approach to model time-stamped security data with the help of knowledge based temporal abstraction to identify suspicious temporal patterns of apps. Based on this, further, they proposed Andromaly [24] for behavior based detection of malware for Android smartphones. Their environment used CPU, memory, power consumption, packets, process related information and other system metrics as features for the static machine learning classifier such as k-means, Logistic regression (LR), Histograms, Decision tree, Bayesian networks and Naive Bayes. The authors reported, Andromaly can detect attacks such as denial of service (DOS), Information theft attacks, Trojan horse attacks etc. To perform DOS and information theft attacks, the authors had created Tip calculator and Schedule SMS, Lunar Lander, Snake, and hypertext transfer protocol (HTTP) upload malicious apps for their research. This helped them to state the performance of their malware detection system. However, they did not test it against the real-world malicious apps. Overall, they discussed a detailed analysis of various machine learning classifiers and feature selection methods. In [25] proposed complementary technique called as TCADS - Trustworthy, Context-related Anomaly Detection for Smartphones that used the similar set of features of Andromaly with the trust and context-related information. Though TCADS itself showed a reasonable effective security mechanism for Android malicious apps, it completely failed when it was tested on real world environment. In [26] used similar features as of Andromaly to identify the malicious activities in cloud infrastructure of mobile devices. In [27] used combination of both the system calls and similar features of Andromaly.

All of the above mentioned studies were done only on small number of benign and malware apps. In [6] had showed large-scale analysis of machine learning classifiers on more number of real apps. In addition, the authors proposed STREAM (System for Automatically Training and Evaluating Android

Malware Classifiers) environment for collection of feature vector. The package runs adb (Android debug bridge) scripts in Linux OS to conduct automated analysis of benign and malware apps. The detailed description about data is discussed in Section 4.1. To continue further research, STREAM package provides dataset (both training and testing feature vectors) in ARFF (Attribute-Relation File Format) format. Using WEKA tool, the authors were able to achieve accuracy in the range 70–81.25% with the static machine learning classifiers namely, Random forest, NB, Multilayer Perceptron (MLP), Bayes net, LR, and J48. They reported 81.25% malware detection rate by using Bayes net. This paper extends the research work of [6] by employing long short-term memory (LSTM).

Deep learning has largely been used for the tasks in speech recognition, image classification, and natural language processing. Towards deep learning based Android malware detection, Droid-Sec [28] tool was introduced. Authors reported deep learning methods are suitable for Android malware classification in comparison to static machine learning classifier with total 202 features from 3 categories such as permission, sensitive API and dynamic behavior in static and dynamic analysis. The results showed 96.5% accuracy using DNN (deep neural network) in real Android malware detection task. Further, they proposed DroidDetector [29] based on the previous research and used static machine learning and deep learning classifier with the benign data set from 20000 benign apps and 1760 malware apps from Contagio Community and Genome Project. It achieved 96.76% as the highest accuracy using deep learning with 192 features in static and dynamic analysis.

3. Background

3.1. Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is similar type of feed-forward network (FFN) with an additional self-recurrent connection that carries out the information from one time-step to the others [30]. Moreover, computational unit of RNN (Fig. 1) remains same as FFN except the connections among neurons in RNN architecture are entirely different. They are connectionist model that have capability to learn dynamic temporal patterns in sequences of arbitrary length. RNN remained as a baseline approach towards the many recent RNN variants model in deep learning.

In general, the extracted feature sequences of benign and malicious apps of arbitrary length are given as input sequence $x = (x_1, x_2, ..., x_T)$ to standard RNN network that estimates the hidden vector sequence $hi = (hi_1, hi_2, ..., hi_T)$ and the output vector sequence $ot = (ot_1, ot_2, ..., ot_T)$ for $t = 1$ to T via the following recursive equations.

$$Hi(x, h) = S(w_{xhi}x + w_{hihi}hi + b) \tag{1}$$

$Hi : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}^q$, $S : \mathbb{R} \to \mathbb{R}$, $w_{xhi} \in \mathbb{R}^{q\times p}$, $w_{hihi} \in \mathbb{R}^{q\times q}$ and $b \in \mathbb{R}^q$, where, S is a hidden layer recurrent activation function such as *sigmoid*, b denotes bias vector, w_{xhi} is input-hidden weight matrix, w_{hihi} is the hidden-hidden weight matrix, w_{hiot} is the hidden-output weight matrix and x denotes input sequences.

The initial step $hi0 = 0$ is used as input to the next layer $hi_1 = f(x_1, hi_0)$ and recursively computes $hi_T = f(x_T, hi_{T-1})$. The hi_{T-1} will help to remember the past information and that will be used

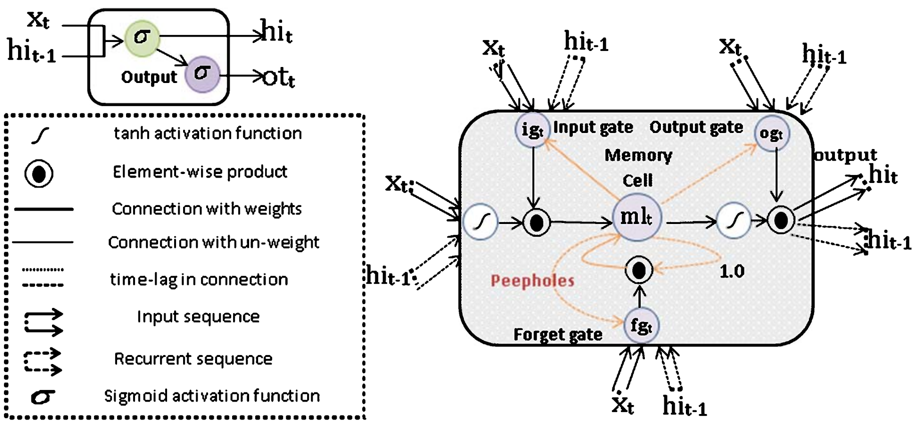

Fig. 1. Architecture of RNN cell and LSTM memory block adopted from [16, 32, 39].

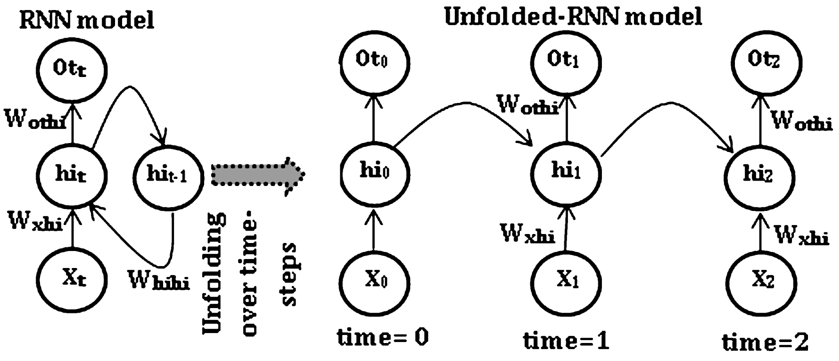

Fig. 2. Unfolding RNN across time steps $t = 0, t = 1, t = 2$.

in calculation of the present units. This hi_T can be further fed to either the stacked recurrent layer or the final layer in which a layer has non-linear activation function, namely *sigmoid* function (sg) as defined below.

$$ot_t = sg(w_{othi}hi_t + b_{ot}) \tag{2}$$

3.1.1. Training in RNN

Training RNN requires the network to be transformed to FFN using unfolding or unrolling. The unfolded RNNs can be interpreted as the deep FFNs without cyclic connections. Deep FFNs usually consist of k hidden layers with the input sequence of length k, as shown in Fig. 2. The newly formed FFNs are flexible for back-propagation. However, the network parameters in RNN are shared across all time-steps. Hence, to estimate the gradient at a particular time-step, rely on the current and as well as previous time-steps. This strategy is called as back-propagation through time (BPTT). In fact, RNN has shown remarkable performance on different tasks such as speech recognition, computer vision and natural language processing areas [31]. However, it led to vanishing and exploding gradient problem while training RNN to learn long-term temporal dynamics of sequences of arbitrary length across many time-steps [32]. This is fundamentally due to the fact that gradient vector can grow or decay exponentially when propagating through many layers of RNN to learn long-term dependencies. Gradient clipping is one of the solutions to reduce the exploding gradients [40]. Vanishing issue can be alleviated with correct initialization of weight matrix w and usage of $ReLU$ activation function instead of $tanh$. Further research brought out many variants to the RNN. In that, LSTM [32] emerged as a successful paradigm to handle long-term dependencies of sequences of arbitrary length.

3.2. Long Short-term Memory (LSTM)

Long short-term memory (LSTM) is a modified network of RNN proposed to learn long-range dependencies across time-varying patterns. Generally, LSTM is a second order recurrent neural network that solves the vanishing and exploding gradients issue by replacing RNN simple units with the memory blocks in recurrent hidden layer. A memory block is a complex processing unit in LSTM with many units (Fig. 1). It is composed of one or many memory cell, adaptive multiplicative gating units (input, output and forget) and a self-recurrent connection with a fixed weight 1.0. It serves as a short-term memory with a control from adaptive multiplicative gating units. The input and output flow of a cell activation of a memory cell is controlled by input and output gate respectively. Forget gate was included in memory cell [16] that helps to forget or reset their previous state information, when it is inappropriate. Moreover, peephole connections between cells to all of its adaptive multiplicative gates control the precise timing of outputs including the internal states [39].

In general, the sequence of features $x = (x_1, x_2, ..., x_T)$ of Android apps are passed as input to LSTM and that estimates an output sequence $ot = (ot_1, ot_2, ..., ot_T)$ with continuous estimation of the gates such as input gate (ig), output gate (og), forget gate (fg) and updating a memory cell (ml) activations in an iterate manner from $t = 1$ to T. The computation of recurrent hidden layer function at time-step T can be generally formulated as follows,

$$\begin{aligned} &x_t, hi_{t-1}, ml_{t-1} \rightarrow hi_t, ml_t \\ &ig_t = sigm(w_{xig}x_t + w_{hiig}hi_{t-1} \\ &\qquad + w_{mlig}ml_{t-1} + b_{ig}) \end{aligned} \tag{3}$$

$$fg_t = sigm(w_{xfg}x_t + w_{hifg}hi_{t-1} + w_{fgml}ml_{t-1} + b_{fg}) \quad (4)$$

$$ml_t = fg_t \odot ml_{t-1} + in_t \odot \tanh(w_{mlx}x_t + w_{mlhi}hi_{t-1} + b_{ml}) \quad (5)$$

$$ot_n = sigm(w_{otx}x_t + w_{othi}hi_{t-1} + w_{otml}ml_t + b_{ot}) \quad (6)$$

$$hi_t = ot_t \odot \tanh(ml_t) \quad (7)$$

where *sigm* is the element-wise *sigmoid* non-linear activation function with range of [0, 1], *tanh* is the element-wise hyperbolic tangent non-linear activation function that has results in the range of [−1,1], $\odot$ is the element-wise multiplication of vectors, the w terms denotes the weight matrices, $b_{in}, b_{fg}, b_{ml}, b_{ot}$ are bias units of input gate, forget gate, memory cell and output gate respectively and ml_t directs LSTM network to learn long-term temporal dynamics in sequence data by proactively selecting either the present input using input gate ig_t or forgetting the previous stated information using fg_t.

4. Methodology

4.1. Description of feature collection in Android

Android is essentially a version/derivative of Linux OS. It is largely adopted in mobile devices, smartphones and tablets, currently accounting for more than a billion of end-users [33]. In addition, it is also preferred as an OS that powers majority of TVs, cars, embedded and wearable devices with each of them having their own apps that are running on the Android system. Android OS primarily consists of four layers in its stack such as Linux kernel, libraries, application framework and apps. The Linux kernel assigns a unique user-id for each app to isolate from other app. The application layer access permission to other apps should be explicitly specified by the application in other layers such as libraries, application framework and apps. So the libraries, application framework and apps don't have any privileges itself, unless defined by the apps. Android features can be collected through rooted or un-rooted devices [7]. The collected features are used to create a model using supervised and unsupervised machine learning classifiers with the aim to detect and classify the malicious apps in Android devices.

4.1.1. Description of Android malware data set for static analysis

For the experiment and evaluations, the recently released data set from [17] is used for static analysis. The authors have collected 279 low-privileged apps and 279 malicious apps from MalGenome. Using ApkTool 2.0.3 obtained AndroidManifest.xml for 558 APK. Each APK outputs granular permissions of feature vector that consist of 330 permissions with a class label as either benign or malign. This can be mathematically defined as, $V = (p_1, p_2, \cdots p_{330}, c)$, where p denotes the permission and c denotes a class label. The values of c are either 1 as granted permissions or 0 as not granted permissions.

4.1.2. Description of Android malware data set for dynamic analysis

The popular data set from [6] is used for evaluating experiments in dynamic analysis. In order to analyze .apk Android files, the authors used shell scripts with "adb-monkey" tool in emulators to generate random user events in communicating with the application interface. STREAM is a feature collection platform proposed by them that facilitates profiling apps and subsequently to train a model using machine learning classifiers. Authors collected feature vectors of battery, binder, memory and permissions by sending more number of emulated user input events to 1330 malicious apps and 408 benign apps at 5 seconds time-interval (totally collected 6,832 feature vectors). The collected binder and battery information based on intent or ActionListenr event, Linux 'top' command execution gave CPU information, and primary Linux system of Android 'Proc' directory gave memory information. In total, the data set have 42 feature dimension with 6,832 feature vectors for training and 306 feature vectors for testing. Each feature vector is of the form $V = (f_1, f_2, \cdots f_{42}, c)$, where f denotes the features and c denotes a class label. The values of c are either 1 as malware or 0 as benign. Different machine learning classifiers on these training and testing data sets is performed and classified 85% of feature vector in training and 70% feature vectors in testing. They were able to achieve high accuracy for different configurations of cross-validation with 6 different machine learning classifiers. [34] author reported accuracy as 97.10% on custom built-in data set of [6]. They used totally 1153 feature vectors. These 1153 feature vectors are randomly divided in to 807 feature vectors for training and 346 feature vectors for testing.

4.2. *Experiments of LSTM-RNN*

All experiments are run on Graphics processing unit (GPU) enabled TensorFlow (r0.11.0) [35] computational framework in Ubuntu 14.04 in conjunction with Keras[1]. LSTM network consist of a set of parameters such as learning rate, memory blocks, number of hidden layer, the number of epochs etc. To choose a good parameter value, the various configurations of network structure and parameters of LSTM are used in each experiment. BPTT technique is used in training the LSTM model and the memory cells of LSTM used as input and output squashing function, *sigmoid* for gates.

4.2.1. *Hyper parameters tuning of LSTM network*

With the aim to identify the appropriate parameters for LSTM network, initially started to experiment with the moderately-sized LSTM. It composed of 3 layers, (1) input layer: contains 27 neurons for minimal feature sets of [34], in static analysis 331 neurons, 42 neurons in dynamic analysis, (2) hidden layer: consist of four memory blocks and (3) output layer has two neurons, one for benign and other for malicious. The connections between input-to-hidden and hidden-to-output are fully connected.

Initially, all instances are mapped in to the range [0–1]. We used all features for training LSTM with various configurations of parameters and structure. While training, epoch parameter value for LSTM is set to 100. To evaluate the performance of LSTM model, confusion-matrix is studied. To know the ideal parameters for network parameters and network structure, the LSTM model is trained for static analysis, dynamic analysis and evaluated it on the test data. The type of networks experimented during training are (1) single LSTM layer (2) stacked LSTM. Four trails for each experiment are run with varied LSTM blocks from 4 to 64, learning rate in the interval [0.01–0.5] and ran up to 500 epochs. The observed results from the various experiments shows that the lower learning rate with more number of LSTM memory blocks in hidden layer performed well with better classification results. In particular, improved performance is achieved by using a lower learning rate i.e. 0.1 with a single LSTM layer with 64 memory blocks. This architecture is able to classify all benign apps successfully within a few numbers of epochs specifically in the range [100–150] for dynamic analysis. However, in some cases the malicious apps are misclassified as benign. For malicious apps, the network required more iteration to learn the entire behavioral patterns, particularly in the range [800–900]. The malware detection rate with varied learning rate in the range [0.01–0.5] for dynamic analysis is shown in Fig. 3(a). For static analysis, Single LSTM layer network performed poorly when compared to dynamic analysis as shown in Fig. 3(b). The network identified benign apps is relatively good at learning rate 0.1 and the detection rate reduced gradually started thereafter. This is due to the overfitting in model training thereby misclassifying some malware apps as benign. Hence, decided to use learning rate 0.1 for both the static and dynamic analysis with the aim to provide a balance between the training time and classification performance.

To enhance the detection rate in static analysis, the concept of deep or stacking an LSTM layer network structure is used. The stacked LSTM network is constructed by placing a stack of LSTM layer on top of each other. These LSTM layers are fully connected and recurrent to it. Each lower LSTM layer hi_t serves its output as an input x_t to the next upper LSTM layer hi_{t+1}. Overall, stacked LSTM learns the long-range dependencies across time-step by integrating multiple levels of representation of inputs through performing more non-linear operations. The number of LSTM hidden layer performance with accuracy is shown in Fig. 3(c).

4.2.2. *LSTM - RNN network topology*

To find an appropriate LSTM network structure, two different network topologies with varying network parameters and network structures are used for both static and dynamic Android malware detection. (1) Single LSTM layer with 32 memory blocks (2) Stacked LSTM layer with 32 memory blocks. Both the network topology is run for 4 trails up to 1,000 epochs. To balance between the model performance on test data and the time for training to create a model, the learning rate is fixed as 0.1 for all network topology experiments. The LSTM network with single layer having 32 memory blocks achieved a good detection rate in terms of accuracy at 300–500 epochs. The malware classes required more than 200–300 epochs to attain better accuracy. In the lesser epochs range, some malwares are classified as benign. To avoid this, LSTM network is run up to 1000 epochs. After 400–500 epochs, the network started to overfit, as shown in Fig. 4(a). In static analysis, to improve the detection rate of malwares,

[1]https://keras.io/

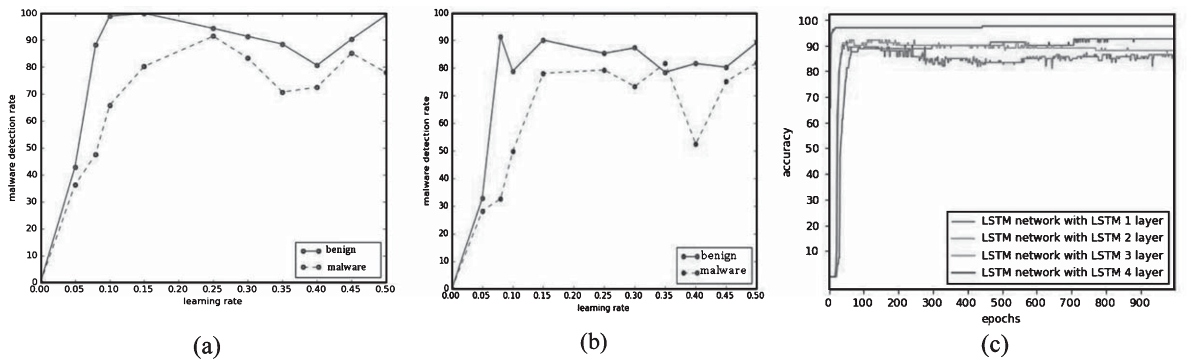

Fig. 3. Performance of (a) Dynamic and (b) Static analysis across learning rate in the range [0.01–0.5], (c) Android malware detection rates in static analysis using LSTM network structures.

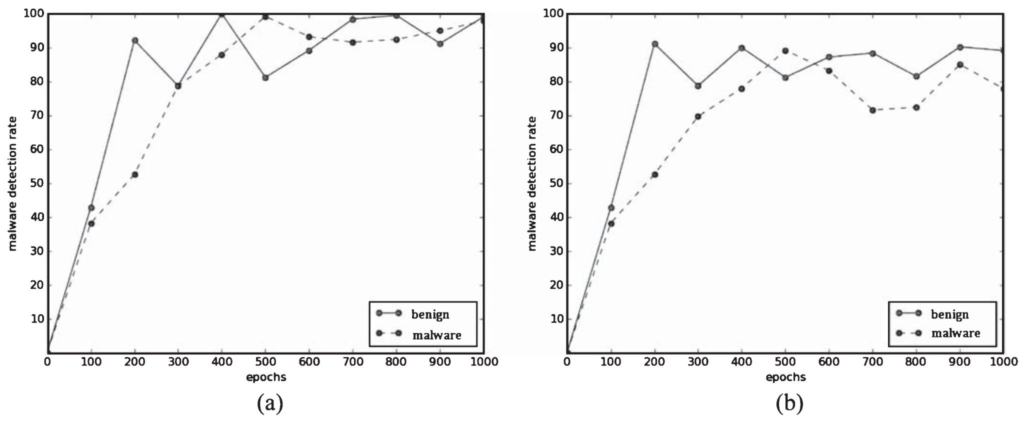

Fig. 4. (a) LSTM network structure performance of (a) dynamic and (b) static analysis.

the memory blocks in LSTM are increased from the range [4–64] to [64–256]. Consequently, the LSTM network followed better malware detection rates. Instead of increasing the memory blocks in a single layer, stacked layer is used with the same model parameters. Hence, in each time-step the input passes through more non-linear operations. Stacked LSTM layer with 32 memory blocks took large number of epochs to achieve good performance in terms of malware detection rate, as shown in Fig. 4(b). This showed a slightly better malware detection rate with fixed learning rate 0.1. In addition to that, this architecture required very large number of epochs to find out malware app.

4.2.3. *Architecture of Long Short-term Memory*

Two LSTM networks are used, (1) A standard LSTM network composed of input layer, LSTM hidden recurrent layer and an output layer, a subset of this is shown in Fig. 5(a). (2) Stacked LSTM network composed of input layer, 3 LSTM hidden layers and output layer, as shown in Fig. 5(b). For both the networks, the binary_crossentropy as loss function, *adam* as optimizer and *sigmoid* as an activation function in output layer are used. A standard LSTM network consists of sequences of multiplication and summation units with adding a connection between them. The network has 42 input neurons (each neuron for unique input feature), the connection between each input neurons to each memory blocks in hidden layer and each memory blocks in LSTM hidden layer to output neurons are fully connected.

Usage of number of memory blocks in a LSTM hidden layer depends on the complexity of data. The optimal number of memory blocks can be chosen by following a parameter tuning at training a LSTM model, as discussed in Section 4.2.1. Each memory cell output in a memory block is fully connected to all gates and output layer neurons. These gates avoid the state of vanishing and exploding gradient issues by carrying out information over long periods of time

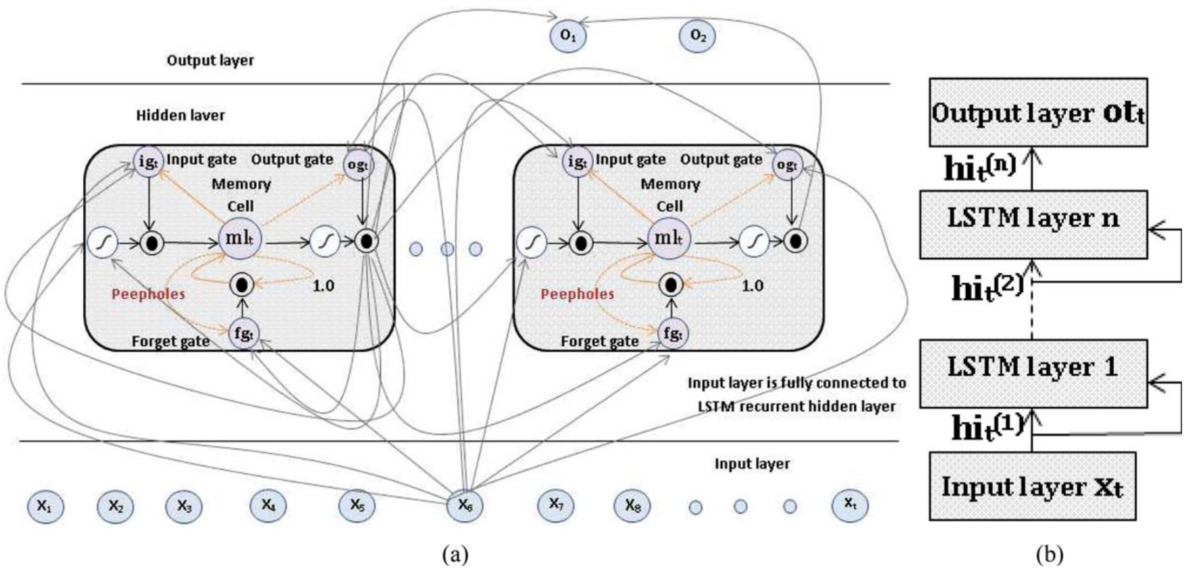

Fig. 5. (a) LSTM three layers network stack (b) stacked LSTM network.

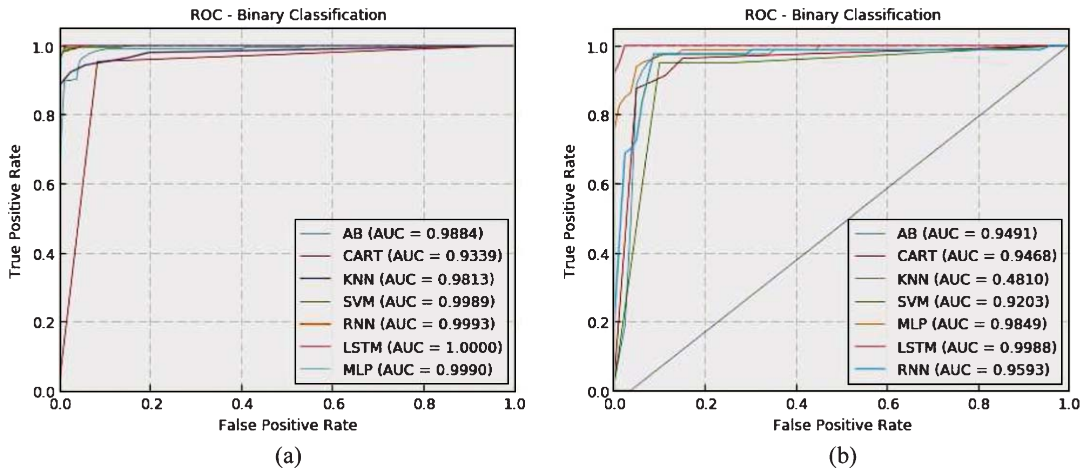

Fig. 6. (a) ROC curve for dynamic analysis, (b) static analysis.

to a memory cell with the continuous updating the state of current memory cell and hidden state. The input and output gate controls the input and output flow of a cell activation of a memory cell at each time steps. The previous state of each time-step is controlled by forget gate. The LSTM network follows a unidirectional flow of information from an input layer to a LSTM hidden layer and a LSTM hidden layer to units in an output layer.

The performances of various classical and deep learning algorithms for both dynamic and static are displayed in receiver operating characteristic (ROC) curve in Fig. 6. In dynamic analysis, LSTM has showed good performance (AUC of 1.000) including the consistent TPR and FPR and static analysis LSTM has showed good performance (AUC of 0.998) including the consistent TPR and FPR.

5. Evaluation results

From Table 2, we can deduce that deep learning mechanisms outperform the other static machine learning classifiers.

Generally, the given input passes through several hidden layers with activation in each hidden layer open the door to differentiate either the app is benign or malicious. Each hidden layer includes

Table 1
Summary of test results of minimal feature sets using LSTM and RNN

Feature sets	LSTM				RNN			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Battery & permission	0.814	0.708	0.969	0.818	0.788	0.677	0.971	0.798
Binder & permission	0.947	0.942	0.935	0.938	0.937	0.955	0.894	0.924
Memory, CPU & permission	0.871	0.867	0.957	0.910	0.858	0.837	0.983	0.904
Network & permission	0.848	0.989	0.821	0.897	0.799	0.690	0.969	0.806

Table 2
Summary of test results for dynamic and static analysis

Algoritm	Dynamic				Static			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
KNN	0.942	0.953	0.943	0.948	0.875	0.866	0.887	0.877
CART	0.934	0.929	0.953	0.933	0.90	0.890	0.912	0.901
Ada boost	0.954	0.954	0.964	0.959	0.919	0.894	0.950	0.921
SVM	0.980	0.989	0.974	0.982	0.925	0.905	0.950	0.925
MLP	0.983	0.990	0.979	0.984	0.938	0.917	0.963	0.939
RNN	0.986	0.990	0.984	0.987	0.944	0.918	0.975	0.945
LSTM	0.997	1.0	0.995	0.997	0.975	0.952	1.0	0.976

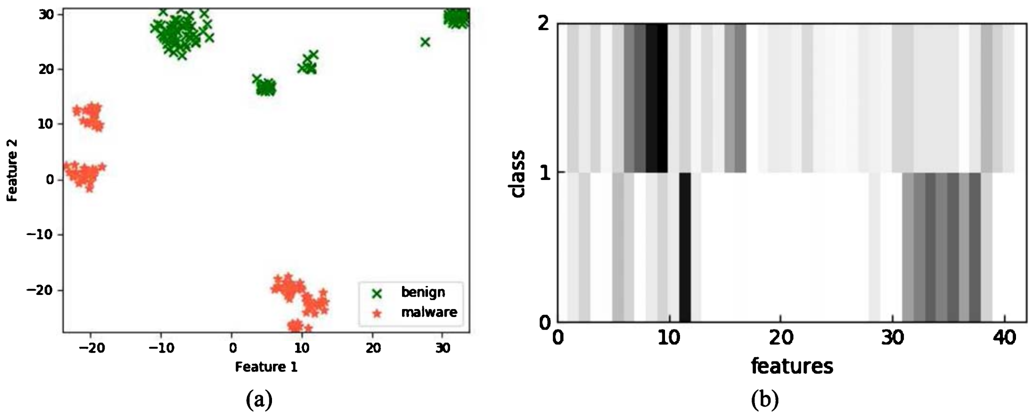

Fig. 7. (a) t-SNE visualization, (b) Saliency map for a malicious application.

a large amount of hidden information that represents the feature representation of the input samples. This can be easily seen by unwrapping them. To do this, we choose 100 samples from benign and malicious classes. Using the trained LSTM models, during testing we fed those 100 samples to LSTM architecture. And instead of passing the high-dimensional hidden layer feature representation to *sigmoid* layer for classification, we redirected in to t-SNE [36]. t-SNE is a dimensionality reduction mechanism that reduced the high-dimensional hidden layer feature representation into two-dimensional representation. Those two-dimensional representations are plotted and as shown in Fig. 7(a). This shows that the apps with similar characteristics are clustered together. Moreover, there is a clear separation between benign and malware classes. This infers that the LSTM model has learned appropriate feature representation that can achieve high malware detection rate.

5.1. *Performance evaluation using minimal feature sets*

To analyze the effectiveness of each individual feature sets for the detection of malicious Android apps, the data set of [6] are divided into different sections [34]. These various sections of minimal features sets are used in training and testing the RNN and its variant, LSTM networks. Various network configurations and its topologies of LSTM and RNN network are trained for each minimal feature sets with learning rate 0.1. All the experiments are run till 500 epochs. The detailed results is displayed in Table 1.

Table 3
Test results of the RNN and LSTM network

Architecture	Accuracy	Precision	Recall	F1-score
LSTM 1 layer	0.767	0.767	1.000	0.868
LSTM 2 layer	0.770	0.770	0.999	0.869
LSTM 3 layer	0.775	0.773	1.000	0.872
LSTM 4 layer	0.872	0.907	0.929	0.918
LSTM 6 layer	**0.939**	**0.937**	**0.987**	**0.961**
RNN 1 layer	0.719	0.651	0.740	0.693
RNN 2 layer	0.767	0.767	1.000	0.868
RNN 3 layer	0.767	0.767	1.000	0.868
RNN 4 layer	0.871	0.863	0.988	0.921
RNN 6 layer	**0.874**	**0.859**	**1.000**	**0.924**

5.2. *Evaluation results for entire data set*

In the last experiment, using the knowledge from the past experiment analysis, the value of learning rate is set to 0.1 and the number of memory blocks to 32. The various configurations of network topologies are constructed and 4 trails of experiments on network topologies run till 1000 epochs to measure the efficacy of deep networks on the entire full data of [6]. For validation purpose, we randomly selected 10% train data. The performances of various network topologies are placed in Table 3.

In order to understand the contribution of each feature in classification process, we adopt back-propogation strategy proposed by [37] in computer vision. The input sample belongs to malicious is chosen and fed to LSTM network. We used Taylor expansion and estimated the first order partial derivative of the neuron activation of last layer (before dense layer with *sigmoid* non-linear activation function for obtaining class probability values) in LSTM architecture. At last, the resultant vectors are plotted as shown in Fig. 7(b). This infers that the features of network have similarity in both benign and malicious. Moreover, the features belong to binder contributed much towards the direction of classifying the given sample as malicious.

6. Conclusion and Future work

This paper analyzes the effectiveness of recurrent neural network (RNN) and its variant long short-term memory (LSTM) for Android malware detection of time-varying sequences of benign and malware apps. In addition, for comparative analysis the other static machine learning classifiers are examined. LSTM network showed superior performance in comparison to RNN and the other strong static machine learning classifiers in assessing the representative data sets of static and dynamic analysis strategies through various empirical experiments. This is primarily due to the fact that the LSTM has the potential to store long-range dependencies across time-steps and to correlate with successive connection sequences information. Overall deep learning approaches, particularly LSTM and its variants are performed well in identifying the hidden malware patterns in a large number of apps.

Though, LSTM network have attained remarkable results on the current data sets, still there is a room for future enhancement. Our future work is of two-fold. One is focused on applying the discussed LSTM network topologies on real raw Android malware samples instead of feature vector of granular permissions in static analysis and profiled application features in dynamic analysis. Another is to focus on studying the internal mechanics of a memory block in each and every time-step of LSTM as it is giving better results. One way to achieve this is to transform states in LSTM network to linear form and from that calculate the eigenvalue and eigenvector to know which eigenvector is actually carrying out required application information from one time-step to the others [38]. The same presented methods could be applied in other security tasks such as PC's malware detection, binary analysis and malicious code detection. Furthermore, the proposed method can serve as an experimental baseline for further research in Android malware detection through various deep learning algorithms.

Acknowledgments

This research was supported in part by Paramount Computer Systems. We are also grateful to NVIDIA India, for the GPU hardware support to research grant.

References

[1] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen and C. Platzer, Andrubis-1,000,000 Apps Later: A view on current Android malware behaviors, *Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, IEEE 2014, pp. 3–17.

[2] CISCO, Cisco 2014 annual security report, http://www.cisco.com/web/offer/gistty2asset/Cisco2014ASR.pdf, 2014.

[3] Android under Siege: Popularity comes at a price, Trend-Labs 3Q 2012 Security Roundup. 2012.

[4] J. Oberheide and C. Miller, Dissecting the Android bouncer, SummerCon2012, New York, 2012.

[5] R.A. Botha, S.M. Furnell and N.L. Clarke, From desktop to mobile: Examining the security experience, *Computer & Security* **28**, 130–137.

[6] B. Amos, H. Turner and J. White, Applying machine learning classifiers to dynamic Android malware detection at scale, *9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2013, pp. 1666–1671.

[7] P. Szor and P. Ferrie, Hunting for metamorphic, *Virus* (2001), 123–143.

[8] Malware detection Methods, Retrieved October 25, 2016, from http://www.avg.com/us-en/avg-software-technology.

[9] P.K. Chan and R.P. Lippmann, Machine learning for computer security, *Journal of Machine Learning Research* **7**, 2669–2672.

[10] C. Metz, Google's training its ai to be Android's security guard. Wired, Retrieved from: https://www.wired.com/2016/06/googles-Android-security-machine-learning.

[11] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature* **521** (2015), 436–444.

[12] R.C. Staudemeyer, Applying long short-term memory recurrent neural networks to intrusion detection, *In South African Computer Journal* **56**(1) (2015), 136–154.

[13] E.C.R. Shin, D. Song and R. Moazzezi, Recognizing Functions in Binaries with Neural Networks, *In Proceedings of the 24th USENIX Security Symposium (USENIX Sec'15)*, 2015, pp. 611–626.

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

[15] Z. Wang, *The application of deep learning on traffic identification*, BlackHat USA, 2015.

[16] F.A. Gers, J. Schmidhuber and F. Cummins, Learning to forget: Continual prediction with LSTM, 1999, pp. 850–855.

[17] L.C.C. Urcuqui and A.N. Cadavid, Framework for malware analysis in Android, *Sistemas & Telemática* **14**(37), 45–56.

[18] J. Sahs and L. Khan, A machine learning approach to Android malware detection, *In Intelligence and Security Informatics Conference (EISIC), 2012 European*, 2012, pp. 141–147. doi: 10.1109/EISIC.2012.34

[19] N. Peiravian and X. Zhu, Machine learning for Android malware detection using permission and api calls, *In Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, 2013, pp. 300–305.

[20] L. Xie, X. Zhang, J. Seifert and S. Zhu, pbmds: A behaviorbased malware detection system for cellphone devices, *in Proceedings of the third ACM Conference on Wireless Network Security ACM*, 2010, pp. 37–48.

[21] T. Isohara, K. Takemori and A. Kubota, Kernel-based behavior analysis for Android malware detection, *in Computational Intelligence and Security (CIS), 2011 Seventh International Conference on IEEE*, 2011, pp. 1011–1015.

[22] M. Egele, T. Scholte, E. Kirda and C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, *ACM Comput Surv* **44**(2) (2012), 1–42.

[23] A. Shabtai, U. Kanonov and Y. Elovici, Intrusion detection on mobile devices using the knowledge based temporalabstraction method, *Journal of Systems and Software* **83**(8), 1524–1537.

[24] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, Andromaly: A behavioral malware detection framework for Android devices, *Journal of Intelligent Information Systems* **38**(1) (2011), 161–190.

[25] I. Bente, B. Hellmann, J. Vieweg, J.V. Helden and G. Dreo, TCADS: Trustworthy, Context-Related Anomaly Detection for Smartphones, *2013 16th International Conference on Network-Based Information Systems*, 2012, pp. 247–254. doi: 10.1109/NBiS.2012.53

[26] T. Kim, Y. Choi, S. Han, J. Chung, J. Hyun, J. Li and J. Hong, Monitoring and detecting abnormal behavior in mobile cloud infrastructure, *in Network Operations and Management Symposium (NOMS), 2012 IEEE IEEE*, 2012, pp. 1303–1310.

[27] G. Dini, F. Martinelli, A. Saracino and D. Sgandurra, MADAM: A multilevel anomaly detector for Android malware, *In International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, Springer Berlin Heidelberg, 2012, pp. 240–253.

[28] Z. Yuan, Y. Lu, Z. Wang and Y. Xue, Droid-sec: Deep learning in Android malware detection, *in Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM, poster)*, 2014, pp. 371–372.

[29] Z. Yuan, Y. Lu and Y. Xue, Droiddetector: Android malware characterization and detection using deep learning, *Tsinghua Science and Technology* **21**(01) (2016), 114–123.

[30] J.L. Elman, Finding structure in time, *Cognitive Science* **14**(2) (1990), 179–211.

[31] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature* **521** (2015), 436–444.

[32] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation* **9**(8) (1997), 1735–1780.

[33] S. Pichai, Google I/O 2014 - Keynote [video. 6:43m], Retrieved from https://www.google.com/events/io

[34] K. Demertzis and L. Iliadis, Bio-inspired hybrid intelligent method for detecting Android malware, *In Knowledge, Information and Creativity Support Systems*, Springer International Publishing, pp. 289–304.

[35] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean and M. Kudlur, TensorFlow: A system for large-scale machine learning, *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, Georgia, USA, 2016.

[36] L. Van der Maaten and G. Hinton, Visualizing data using t-SNE, *Journal of Machine Learning Research* **9**(2579-2605) (2008), 85.

[37] K. Simonyan, A. Vedaldi and A. Zisserman, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, arXiv preprint arXiv:1312.6034, 2013.

[38] R. Moazzezi, Change-based population coding, PhD thesis, UCL (University College London), 2011.

[39] F.A. Gers, N.N. Schraudolph and J. Schmidhuber, Learning precise timing with LSTM recurrent networks, *Journal of Machine Learning Research* **3** (2002), 115–143.

[40] R.J. Williams and D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural Computation, Neural Computation* **1**(2), 270–280.