Bio-inspired Hybrid Intelligent Method for Detecting Android Malware

Konstantinos Demertzis and Lazaros Iliadis

Abstract Today's smartphones are capable of doing much more than the previous generation of mobile phones. However this extended range of capabilities is coming together with some new security risks. Also, mobile platforms often contain small, insecure and less well controlled applications from various single developers. Due to the open usage model of the Android market, malicious applications cannot be avoided completely. Especially pirated applications or multimedia content in popular demand, targeting user groups with typically low awareness levels are predestined to spread too many devices before being identified as malware. Generally malware applications utilizing root exploits to escalate their privileges can inject code and place binaries outside applications storage locations. This paper proposes a novel approach, which uses minimum computational power and resources, to indentify Android malware or malicious applications. It is a bio-inspired Hybrid Intelligent Method for Detecting Android Malware (HIMDAM). This approach performs classification by employing Extreme Learning Machines (ELM) in order to properly label malware applications. At the same time, Evolving Spiking Neural Networks (eSNNs) are used to increase the accuracy and generalization of the entire model.

Keywords Security · Android malware · Evolving spiking neural networks · Extreme learning machines · Radial basis function networks · Polynomial neural networks · Self-Organizing maps · Multilayer perceptron

K. Demertzis (✉) · L. Iliadis (✉)
Department of Forestry and Management of the Environment and Natural Resources, Democritus University of Thrace, 193 Pandazidou St, 68200 N. Orestiada, Greece
e-mail: kdemertz@fmenr.duth.gr

L. Iliadis
e-mail: liliadis@fmenr.duth.gr

S. Kunifuji et al. (eds.), *Knowledge, Information and Creativity Support Systems*, Advances in Intelligent Systems and Computing 416,
DOI 10.1007/978-3-319-27478-2_20

1 Introduction

Lately, the share of smartphones in the sales of handheld mobile communication devices has drastically increased. Among them, the number of Android based smartphones is growing rapidly. They are increasingly used for security critical private and business applications, such as online banking or to access corporate networks. This makes them a very valuable target for an adversary. Until recently, the Android Operating System's security model has succeeded in preventing any significant attacks by malware. This can be attributed to a lack of attack vectors which could be used for self-spreading infections and low sophistication of malicious applications. However, emerging malware deploys advanced attacks on operating system components to assume full device control [10]. Malware are the most common infection method because the malicious code can be packaged and redistributed with popular applications. In Android, each application has an associated *.apk* file which is the executable file format for this platform. Due to the open software installation nature of Android, users are allowed to install any executable file from any application store. This could be from the official Google Play store, or a third party site. This case of installing applications makes Android users vulnerable to malicious applications. Some of the most widely used solutions such as antivirus software are inadequate for use on smartphones as they consume too much CPU and memory and might result in rapid draining of the power source. In addition, most antivirus detection capabilities depend on the existence of an updated malware signature repository, therefore the antivirus users are not protected from zero-day malware.

This research effort aims in the development and application of an innovative, fast and accurate bio-inspired Hybrid Intelligent Method for Detecting Android Malware (HIMDAM). This is achieved by employing Extreme Learning Machines (ELMs) and Evolving Spiking Neural Networks (eSNNs). A RBF Kernel ELM has been employed for malware detection, which offers high learning speed, ease of implementation and minimal human intervention. Also, an eSNN model has been applied to increase the accuracy and generalization of the entire method. In fact, the bio-inspired model has shown better performance when compared to other ANN methods, such as Multilayer Perceptrons (MLP), Radial Basis Function ANN (RBF), Self-Organizing Maps (SOM), Group Methods of Data Handling (GMDH) and Polynomial ANN. A main advantage of HIMDAM is the fact that it reduces overhead and overall analysis time, by classifying malicious and benign applications with high accuracy.

1.1 Literature Review

Significant work has been done in applying machine learning (ML) techniques, using features derived from both static [7, 24, 29] and dynamic [4] analysis to

identify malicious Android applications [13]. Amongst early efforts towards Android applications security was the "*install-time policy security system*" developed by Enck et al. which considered risks associated with combinations of the app permissions [9]. From another perspective, some works focused in the runtime analysis [20, 22] whereas others have tried a static analysis of apps [12]. For instance, Chin et al. [7] used a 2-means clustering [21] of apps' call activities, to detect Trojans. Fuchs et al. [11] used formal static analysis of byte codes [33] to form data flow-permission consistency as a constrained optimization problem. Barrera et al. [3] used app permissions in self-organizing maps (SOMs) to visualize app permission usage as a U-matrix [18]. Besides, their SOM component plane analysis allowed identification of the frequently jointly requested permissions. However, they did not relate categories and permissions. In [30], Tesauro et al. train ANN to detect boot sector viruses, based on byte string trigrams. Schultz et al. [27] compare three machine learning algorithms trained on three features: DLL and system calls made by the program, strings found in the program binary and a raw hexadecimal representation of the binary [23]. Kotler and Maloof [19] used a collection of 1971 benign and 1651 malicious executable files. N-grams were extracted and 500 features were selected using the Information Gain measure. The vector of n-gram features was binary, presenting the presence or absence of a feature in the file. In their experiment, they trained several classifiers: IBK k-Nearest Neighbors (k-NN), a similarity-based classifier called the TFIDF classifier, Naïve Bayes, Support Vector Machines (SVM) and Decision Trees under the algorithm J48 [28]. The last three of these were also boosted. In the experiments, the four best-performing classifiers were Boosted J48, SVM, Boosted SVM and IBK [28]. Also, Cheng et al. [6] proposed the use of ELM methods to classify binary and multi-class network traffic for intrusion detection. The performance of ELM in both binary-class and multi-class scenarios are investigated and compared to SVM based classifiers. Joseph et al., [16] developed an autonomous host-dependent Intrusion Detection System (IDS) for identifying malicious sinking behavior. This system increases the detection accuracy by using cross-layer features to describe a routing behavior. Two ML approaches were exploited towards learning and adjustment to new kind of attack circumstances and network surroundings. ELMs and Fisher Discriminant Analysis (FDA) are utilized collectively to develop better accuracy and quicker speed of method.

2 Methodologies Comprising the Proposed Hybrid Approach

2.1 Extreme Learning Machines (ELM)

The extreme learning machine (ELM) as an emerging learning technique provides efficient unified solutions to generalized feed-forward networks including but not limited to (both single- and multi-hidden-layer) neural networks, radial basis

function (RBF) networks, and kernel learning [34]. ELM theories show that hidden neurons are important but can be randomly generated, independent from applications and that ELMs have both universal approximation and classification capabilities. They also build a direct link between multiple theories namely: ridge regression, optimization, ANN generalization performance, linear system stability and matrix theory. Thus, they have strong potential as a viable alternative technique for large-scale computing and ML. Also ELMs, are biologically inspired, because hidden neurons can be randomly generated independent of training data and application environments, which has recently been confirmed with concrete biological evidences. ELM theories and algorithms argue that "random hidden neurons" capture the essence of some brain learning mechanism as well as the intuitive sense that the efficiency of brain learning need not rely on computing power of neurons. This may somehow hint at possible reasons why brain is more intelligent and effective than computers [5].

ELM works for the "generalized" Single-hidden Layer feedforward Networks (SLFNs) but the hidden layer (or called feature mapping) in ELM need not be tuned.

Such SLFNs include but are not limited to SVMs, polynomial networks, RBFs and the conventional (both single-hidden-layer and multi-hidden-layer) feedforward ANN. Different from the tenet that all the hidden nodes in SLFNs need to be tuned, ELM learning theory shows that the hidden nodes/neurons of generalized feedforward networks needn't be tuned and these hidden nodes/neurons can be randomly generated [34]. All the hidden node parameters are independent from the target functions or the training datasets. ELMs conjecture that this randomness may be true to biological learning in animal brains. Although in theory, all the parameters of ELMs can be analytically determined instead of being tuned, for the sake of efficiency in real applications, the output weights of ELMs may be determined in different ways (with or without iterations, with or without incremental implementations) [34]. According to ELM theory the hidden node/neuron parameters are not only independent of the training data but also of each other. Unlike conventional learning methods which must see the training data before generating the hidden node/neuron parameters, ELMs could randomly generate the hidden node/neuron parameters before seeing the training data. In addition, ELMs can handle non-differentiable activation functions, and do not have issues such as finding a suitable stopping criterion, learning rate, and learning epochs. ELMs have several advantages, ease of use, faster learning speed, higher generalization performance, suitable for many nonlinear activation function and kernel functions [34].

2.2 Evolving Spiking Neural Networks (eSNNs)

The eSNNs are modular connectionist-based systems that evolve their structure and functionality in a continuous, self-organized, on-line, adaptive, interactive way from incoming information. These models use trains of spikes as internal information

representation rather than continuous variables [25]. The eSNN developed and discussed herein is based in the "Thorpe" neural model [31]. This model intensifies the importance of the spikes taking place in an earlier moment, whereas the neural plasticity is used to monitor the learning algorithm by using one-pass learning. In order to classify real-valued data sets, each data sample, is mapped into a sequence of spikes using the Rank Order Population Encoding (ROPE) technique [8, 32]. The topology of the developed eSNN is strictly feed-forward, organized in several layers and weight modification occurs on the connections between the neurons of the existing layers.

The ROPE method is alternative to the conventional rate coding scheme (CRCS). It uses the order of firing neuron's inputs to encode information. This allows the mapping of vectors of real-valued elements into a sequence of spikes. Neurons are organized into neuronal maps which share the same synaptic weights. Whenever the synaptic weight of a neuron is modified, the same modification is applied to the entire population of neurons within the map. Inhibition is also present between each neuronal map. If a neuron spikes, it inhibits all the neurons in the other maps with neighboring positions. This prevents all the neurons from learning the same pattern. When propagating new information, neuronal activity is initially reset to zero. Then, as the propagation goes on, each time one of their inputs fire, neurons are progressively desensitized. This is making neuronal responses dependent upon the relative order of firing of the neuron's afferents [17, 37].

The aim of the one-pass learning method is to create a repository of trained output neurons during the presentation of training samples. After presenting a certain input sample to the network, the corresponding spike train is propagated through the eSNN which may result in the firing of certain output neurons. It is possible that no output neuron is activated and the network remains silent and the classification result is undetermined. If one or more output neurons have emitted a spike, the neuron with the shortest response time among all activated output neurons is determined. The label of this neuron is the classification result for the presented input [26].

3 Description of the Proposed HIMDAM Algorithm

The proposed herein, HIMDAM methodology uses an ELM classification approach to classify malware or benign applications with minimum computational power and time, combined with the eSNN method in order to detect and verify the malicious code. The general algorithm is described below:

Step 1:
Train and test *datasets* are determined and normalized to the interval [−1,1]. The *datasets* are divided in 4 main sectors with "*permission*" feature. *Permission* is a

security mechanism of mobile operating systems. For mobile phones any application executed under the device owner's user ID would be able to access any other application's data. The Android kernel assigns each application its own user ID on installation. To avoid the abuse of mobile phone functions, Android allows the user to effectively identify and manage mobile phone resources by setting permissions. If the application requires a certain function, the developer can announce permission. In the latest version of Android, there are a total of 130 permissions. To malware, some permissions are important and frequently needed, therefore they should be weighted. For example, the attacker needs permissions to transfer the stolen data to his account through the web, or to perform damaging behavior by sending out large number of SMS messages. The features involved can be divided in the sectors below:

1. *Battery + Permissions* (*5 features*)
2. *Binder + Permissions* (*18 features*)
3. *Memory + CPU + Permissions* (*10 features*)
4. *Network + Permissions* (*9 features*)

The *Hardware_Dataset* has been generated (16 features) including the most important variables from hardware related sectors (Battery, Memory, CPU, Network). On the other hand, the *All_Imp_Var_Dataset* (27 features) comprises of the most important variables from all of the sectors (Battery, Memory, CPU, Network, Binder). To calculate the importance of variables we replace them with their mean values one by one and we measure the root mean squared error (RMSE) of the "new" model. Original model error is considered to have a zero percent impact on the RMSE and 100 % impact is a case where all variables are replaced with their mean. The impact can easily exceed 100 % when the variable in a model is multiplied by another one or it is squared. A small negative percentage can also happen if a variable is merely useless for the model.

In order to create a very fast and accurate prediction model with minimum requirements of hardware resources, we randomly check two sectors (e.g. Battery and Binder or Memory and Binder) every time with the ELM classifier. According to the ELM theory [15], the Gaussian Radial Basis Function kernel $K(u,v) = exp(-\gamma||u - v||^2)$ is used. The hidden neurons are k = 20, w_i are the assigned random input weights and b_i the biases, where i = 1,…,N and H is the hidden layer output matrix.

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{bmatrix} \quad (1)$$

$h(x) = [h_1(x), \ldots, h_L(x)]$ is the output (row) vector of the hidden layer with respect to the input x. Function $h(x)$ actually maps the data from the d-dimensional input space to the L-dimensional hidden-layer feature space (ELM feature space) H and thus,

$h(x)$ is indeed a feature mapping. ELM aims to minimize the training error as well as the norm of the output weights as shown in Eq. 2:

$$\text{Minimize: } \|H\beta - T\|^2 \text{ and } \|\beta\| \tag{2}$$

To minimize the norm of the output weights $||\beta||$ is actually to maximize the distance of the separating margins of the two different classes in the ELM feature space $2/||\beta||$.

The calculation of the output weights β is done according to Eq. (3):

$$\beta = \left(\frac{I}{C} + H^T H\right)^{-1} H^T T \tag{3}$$

where c is a positive constant and T is obtained from the *Function Approximation of SLFNs* with additive neurons with $t_i = [t_{i1}, t_{i2},\ldots,t_{im}]^T R^m$ and $T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}$.

It has been shown numerically in ELM theory [15] that the above solution has better generalization performance. More specifically, the reasoning of the new Malware detection algorithm that has been developed in this research is as seen below:

1: ***If*** *both sectors' analysis with the ELM offers Negative results, no action is required and the next 2 sectors are examined.*
2: ***If*** *the ELM analysis results in a Negative result for the one sector and positive for the other then:*
3: ***If*** *both sectors belong to the general Hardware field (eg. Network and Battery) then the Hardware_Dataset is reexamined:*
4: ***If*** *the result is Negative then we go further.*
5: ***If*** *the result is Positive then the whole Original Dataset with all 40 features is checked.*
6: ***If*** *the ELM analysis of both sectors produces Positive results then the whole Original Dataset with all 40 features is checked.*
7: ***If*** *one of sectors belongs to the Binder field then the All_Imp_Var_Dataset is examined:*
8: ***If*** *the result is Negative then we go further.*
9: ***If*** *the result is Positive then the whole Original Dataset with all 40 features is checked with eSNN classification method.*
10: ***If*** *the ELM analysis of both sectors produces Positive result, then the whole Original Dataset is checked with eSNN classifier.*

Step 2:
The train and test datasets are determined and formed, related to n features. The required classes (malware and benign applications) that use the variable *Population Encoding* are imported. This variable controls the conversion of real-valued data

samples into the corresponding time spikes. The encoding is performed with 20 Gaussian receptive fields per variable (Gaussian width parameter beta = 1.5). The data are normalized to the interval [−1,1] and so the coverage of the Gaussians is determined by using i_min and i_max. For the normalization processing the following equation is used:

$$x_{1_{norm}} = 2 * \left(\frac{x_1 - x_{min}}{x_{max} - x_{min}} \right) - 1,\ x \in R \tag{4}$$

The data is classified in two classes namely: ***Class positive*** which contains the benign results and ***Class negative*** which comprises of the malware ones. The eSNN is using modulation factor m = 0.9, firing threshold ratio c = 0.7 and similarity threshold s = 0.6 in agreement with the vQEA algorithm [26, 37]. More precisely, let $A = \{a_1, a_2, a_3 \ldots a_{m-1}, a_m\}$ be the ensemble of afferent neurons of neuron i and $W = \{w_{1,i}, w_{2,i}, w_{3,i} .. w_{m-1,i}, w_{m,i}\}$ the weights of the m corresponding connections; let mod ∈ [0,1] be an arbitrary modulation factor. The activation level of neuron i at time t is given by Eq. 5:

$$\text{Activation}(i, t) = \sum_{j \in [1, m]} \text{mod}^{\text{order}(a_j)}\, w_{j,i} \tag{5}$$

where order(a_j) is the firing rank of neuron a_j in the ensemble A. By convention, order(a_j) = +8 if a neuron a_j is not fired at time t, sets the corresponding term in the above sum to zero. This kind of desensitization function could correspond to a fast shunting inhibition mechanism. When a neuron reaches its threshold, it spikes and inhibits neurons at equivalent positions in the other maps so that only one neuron will respond at any location. Every spike triggers a time based Hebbian-like learning rule that adjusts the synaptic weights. Let t_e be the date of arrival of the Excitatory PostSynaptic Potential (EPSP) at synapse of weight W and t_a the date of discharge of the postsynaptic neuron.

$$\text{If } t_{e<} t_a \text{ then } dW = a(1 - W)e^{-|\Delta o|\tau} \text{ else } dW = -aW\, e^{-|\Delta o|\tau}. \tag{6}$$

Δo is the difference between the date of the EPSP and the date of the neuronal discharge (expressed in term of order of arrival instead of time), a is a constant that controls the amount of synaptic potentiation and depression [8]. ROPE technique with receptive fields, allow the encoding of continuous values. Each input variable is encoded independently by a group of one-dimensional receptive fields (Figs. 1 and 2). For a variable n, an interval $[I^n_{min}, I^n_{max}]$ is defined. The Gaussian receptive field of neuron i is given by its center μ_i and width σ by Eq. 8.

$$\mu i = I^n_{min} + \frac{2i - 3}{2} \frac{I^n_{max} - I^n_{min}}{M - 2} \tag{7}$$

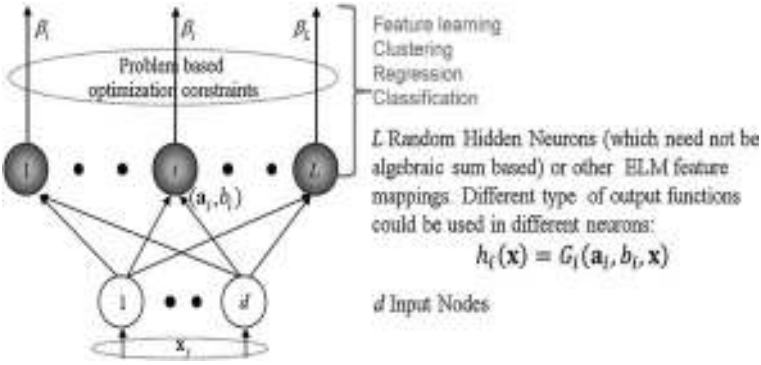

Fig. 1 Extreme learning machine (ELM) [34]

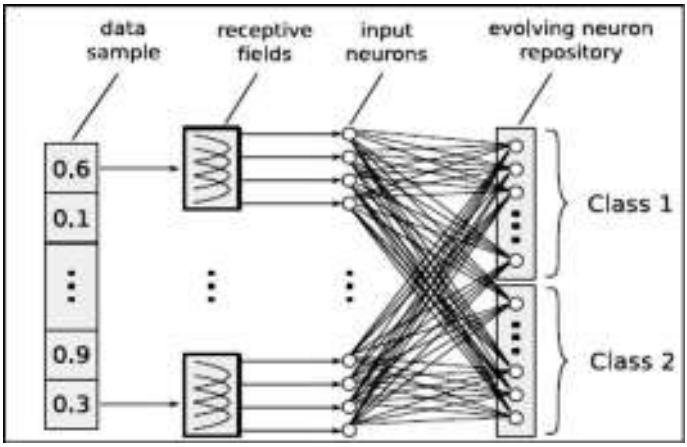

Fig. 2 The Evolving Spiking Neural Network (eSNN) architecture [17]

$$\sigma = \frac{1}{\beta}\frac{I_{max}^{n} - I_{min}^{n}}{M-2} \tag{8}$$

where $1 \leq \beta \leq 2$ and the parameter β directly controls the width of each Gaussian receptive field. Figure 3 depicts an encoding example of a single variable.

For an input value $v = 0.75$ (thick straight line) the intersection points with each Gaussian is computed (triangles), which are in turn translated into spike time delays (right figure) [37].

Step 3:
The eSNN is trained with the *training* dataset vectors and the testing is performed with the *testing* vectors. The procedure of one pass learning is described in the following Algorithm 2 [17, 37].

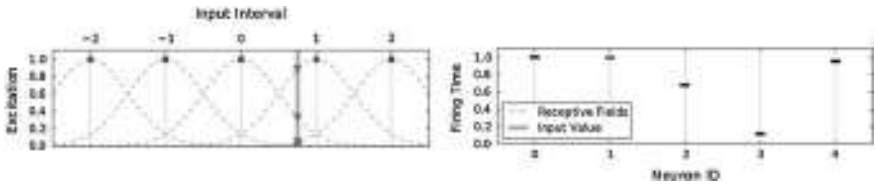

Fig. 3 Population encoding based on Gaussian receptive fields. *Left figure* Input interval—*right figure* neuron ID [17]

Algorithm 1: Training an evolving Spiking Neural Network (eSNN) [37]

Require: m_l, s_l, c_l **for a class label** $l \in L$
1: initialize neuron repository $R_l = \{\}$
2: **for all** samples $X^{(i)}$ belonging to class l **do**
3: $w_j^{(i)} \leftarrow (m_l)^{order(j)}$, $\forall$ j | j pre-synaptic neuron of i
4: $u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)}$
5: $\theta^{(i)} \leftarrow c_l u_{max}^{(i)}$
6: **if** $\min(d(w^{(i)}, w^{(k)})) < s_l$, $w^{(k)} \in R_l$ **then**
7: $w^{(k)} \leftarrow$ merge $w^{(i)}$ and $w^{(k)}$ according to Equation 7
8: $\theta^{(k)} \leftarrow$ merge $\theta^{(i)}$ and $\theta^{(k)}$ according to Equation 8
9: **else**
10: $R_l \leftarrow R_l \cup \{w^{(i)}\}$
11: **end if**
12: **end for**

For each training sample i with class label **l** which represent a benign software, a new output neuron is created and fully connected to the previous layer of neurons, resulting in a real-valued weight vector $w^{(i)}$ *with* $w_j^{(i)} \in R$ denoting the connection between the pre-synaptic neuron j and the created neuron i. In the next step, the input spikes are propagated through the network and the value of weight $w_j^{(i)}$ is computed according to the order of spike transmission through a synapse j: $w_j^{(i)} = (m_l)^{order(j)}$, $\forall j \mid j$ pre-synaptic neuron of i. Parameter m_l is the modulation factor of the Thorpe neural model. Differently labeled output neurons may have different modulation factors m_l. Function order(j) represents the rank of the spike emitted by neuron j. The firing threshold $\theta^{(i)}$ of the created neuron I is defined as the fraction $c_l \in R$, $0 < c_l < 1$, of the maximal possible potential

$$u_{max}^{(i)} : \theta^{(i)} \leftarrow c_l u_{max}^{(i)} \tag{7}$$

$$u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)} \tag{9}$$

The fraction c_l is a parameter of the model and for each class label $l \in L$ a different fraction can be specified. The weight vector of the trained neuron is compared to the weights corresponding to neurons already stored in the repository. Two neurons are considered too "similar" if the minimal *Euclidean* distance between their weight vectors is smaller than a specified similarity threshold s_l (the eSNN object uses optimal similarity threshold $s = 0.6$). All parameters of eSNN (modulation factor m_l, similarity threshold s_l, PSP fraction c_l, $l \in L$) included in this search space, were optimized according to the Versatile Quantum-inspired Evolutionary Algorithm (vQEA) [26]. Both the firing thresholds and the weight vectors were merged according to Eqs. 10 and 11:

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1 + N}, \forall j | j \text{ pre} - \text{synaptic neuron of i} \tag{10}$$

$$\theta^{(k)} \leftarrow \frac{\theta^{(i)} + N\theta^{(k)}}{1 + N} \tag{11}$$

Integer N denotes the number of samples previously used to update neuron k. The merging is implemented as the (running) average of the connection weights, and the (running) average of the two firing thresholds. After merging, the trained neuron i is discarded and the next sample processed. If no other neuron in the repository is similar to the trained neuron i, the neuron i is added to the repository as a new output.

4 Data and Results

For this experiment, we used the free dataset provided by B. Amos [2]. The author developed a shell script to automatically analyze *.apk* Android application files by running them in available Android emulators. For each *.apk* file, the emulator simulates user interaction by randomly interacting with the application interface. This is done using the Android "*adb-monkey"* tool [14]. Based on inspection of the source code, we can conclude that each feature vector of the dataset is collected at 5 s' intervals. The memory features were collected by observing the "*proc*" directory in the underlying Linux system of Android. The CPU information was collected by running the Linux "*top*" command. The Battery and Binder information was collected by using "*intent*" (Action listener) [1].

The original dataset has a total of 1153 data (feature vector) samples with 660 benign samples (classified as positive class) and 493 malicious samples (classified as negative class). It was divided randomly in two parts: 1) a training dataset containing 807 patterns (467 positive and 340 negative patterns) 2) a testing dataset containing 346 patterns (193 positive and 153 negative patterns). To identify the integrity of HIMDAM we have compared the ELM and eSNN classifiers with other neural network methods. The performance of both classifiers was evaluated on

Table 1 Accuracy (ACC) comparison between MLP, RBF, ELM, GMDH PNN, eSNN

	MLP		RBF		SOM		ELM		GMDH PNN		eSNN	
	Acc (%)	Time	Acc (%)	Time	Acc (%)	Time	Acc (%)	Time	Acc (%)	Time	Acc (%)	Time
All features	95.38	38.31	85.72	0.34	87.34	0.20	89.19	**0.17**	90.50	13.00	**97.10**	20.22
Battery + perm	73.58	1.69	71.96	0.19	64.28	0.23	72.49	**0.17**	71.30	2.00	**79.30**	1.22
Binder + perm	93.35	7.95	79.87	0.08	72.82	0.08	82.98	**0.05**	84.80	5.00	**93.60**	8.71
Memory CPU Perm	82.08	4.84	79.65	0.13	77.16	0.23	80.08	**0.12**	83.10	4.00	**84.90**	4.40
Network perm	**81.21**	3.02	69.83	**0.13**	70.10	0.19	71.27	0.14	73.70	3.00	80.10	3.11
Important var. from all features	97.69	14.59	83.76	**0.20**	80.34	**0.20**	89.47	**0.20**	91.00	8.00	**98.20**	11.02
Important var. from hardware	94.22	6.69	88.25	0.22	75.90	0.17	88.00	**0.14**	89.40	6.00	**94.80**	5.91

malware datasets. The results showed that the kernel based ELM has much faster learning speed (run thousands times faster than conventional methods) and the eSNN has much better generalization performance and more accurate and reliable classification results. The comparisons were performed on a dual boot PC with a P4 at 3.1 GHz CPU and 4 GB RAM. For the eSNN classification, the *Linux Ubuntu 12.04 LTS* OS with *PyLab* (*NumPy, SciPy, Matplotlib* and *IPython*) was employed. The MLP, RBF and SOM tests were performed with the *Weka 3.7* [35], ELM with *Matlab 2013* and GMDH PNN with GMDH shell software [36]. The performance comparisons of NN algorithms are shown in Table 1. The confusion matrices for ELM and eSNN can be seen in Table 2 (Fig. 4).

Table 2 Confusion matrices for ELM and eSNN algorithms

Confusion matrices for ELM				Confusion matrices for eSNN			
All features				All features			
	Benign (predicted)	Malware (predicted)	Accuracy (%)		Benign (predicted)	Malware (predicted)	Accuracy (%)
Benign (actual)	180	13	**93.26**	Benign (actual)	190	3	**98.40**
Malware (actual)	23	130	**85.12**	Malware (actual)	7	146	**95.80**
Overall accuracy			**89.19**	Overall accuracy			**97.10**
Battery permissions							
Benign (actual)	149	44	**77.20**	Benign (actual)	160	33	**82.90**
Malware (actual)	50	103	**67.78**	Malware (actual)	37	116	**75.70**
Overall accuracy			**72.49**	Overall accuracy			**79.30**
Binder permissions							
Benign (actual)	164	29	**84,97**	Benign (actual)	185	8	**95.80**
Malware (actual)	29	124	**80,99**	Malware (actual)	13	140	**91.40**
Overall accuracy			**82.98**	Overall accuracy			**93.60**
Memory CPU permissions							
Benign (actual)	159	34	**82.38**	Benign (actual)	168	25	**87.00**
Malware (actual)	34	119	**77.78**	Malware (actual)	26	127	**82.80**
Overall accuracy			**80.08**	Overall accuracy			**84.90**

(continued)

Table 2 (continued)

Confusion matrices for ELM				Confusion matrices for eSNN			
All features				All features			
	Benign (predicted)	Malware (predicted)	Accuracy (%)		Benign (predicted)	Malware (predicted)	Accuracy (%)
Network permissions							
Benign (actual)	142	51	**73.57**	Benign (actual)	161	32	**83.40**
Malware (actual)	48	105	**68.97**	Malware (actual)	36	117	**76.80**
Overall accuracy			**71.27**	Overall accuracy			**80.10**
Importance variables from all							
Benign (actual)	180	13	**93.26**	Benign (Actual)	191	2	**99.00**
Malware (actual)	22	131	**85.68**	Malware (Actual)	4	149	**97.40**
Overall accuracy			**89.47**	Overall Accuracy			**98.20**
Importance variables from hardware							
Benign (actual)	175	23	**90.67**	Benign (actual)	184	9	**95.30**
Malware (actual)	23	130	**85.33**	Malware (actual)	9	144	**94.30**
Overall accuracy			**88.00**	Overall accuracy			**94.80**

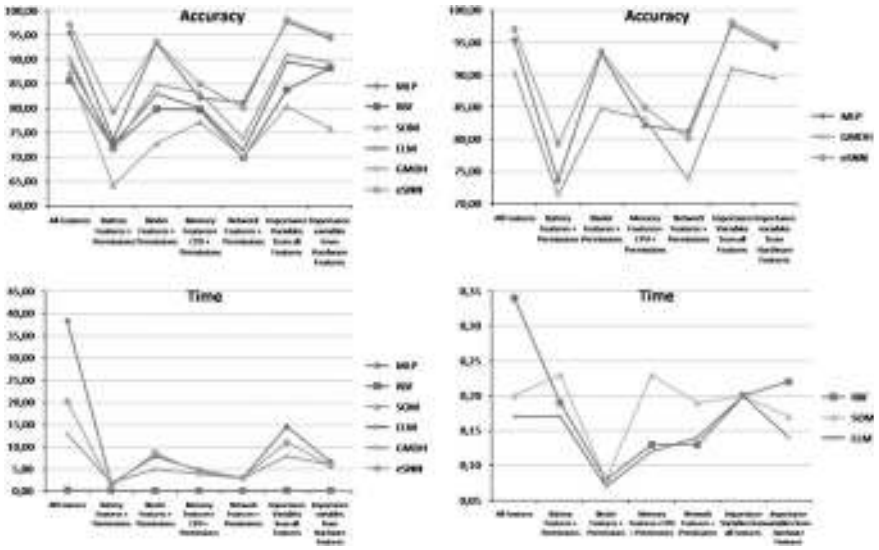

Fig. 4 Accuracy and time comparison of MLP, RBF, ELM, GMDH PNN and eSNN

5 Discussion—Conclusions

An innovative bio-inspired Hybrid Intelligent Method for Detecting Android Malware (HIMDAM) has been introduced in this paper. It performs classification by using ELM (a very fast approach to properly label malicious executables) and eSNN for the detection of malware with high accuracy and generalization. An effort was made to achieve minimum computational power and resources. The classification performance of the ELM and the accuracy of the eSNN model were experimentally explored based on different datasets and reported promising results. Moreover the hybrid model detects the patterns and classifies them with high accuracy. In this way it adds a higher degree of integrity to the rest of the security infrastructure of Android Operating System. As a future direction, aiming to improve the efficiency of biologically realistic ANN for pattern recognition, it would be important to evaluate the eSNN model with ROC analysis and to perform feature minimization in order to achieve minimum processing time. Other coding schemes could be explored and compared on the same security task. Also what is really interesting is a scalability of ELM with other kernels in parallel and distributed computing in a real-time system. Finally the HIMDAM could be improved towards a better online learning with self-modified parameter values.

References

1. Alam M.S., Vuong S.T.: Random forest classification for detecting android malware. In: IEEE IC on Green Computing and Communications and Internet of Things (2013)
2. Amos, B.: Antimalware. https://github.com/VT-Magnum-Research/antimalware (2013)
3. Barrera, D., Kayacik, H., Oorshot, P., Somayaji, A.: A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. ACM (2010)
4. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: 1st ACM Workshop on on SPSM, pp. 15–26. ACM (2011)
5. Cambria E., Huang G.-B.: Extreme learning machines. IEEE Intell. Syst. (2013)
6. Cheng, C., Peng, W.T, Huang, G.-B.: Extreme learning machines for intrusion detection. In: WCCI IEEE World Congress on Computational Intelligence Brisbane, Australia (2012)
7. Chin E., Felt A., Greenwood K., Wagner D.: Analyzing inter-application communication in android. In: 9th Conference on Mobile Systems, Applications, and Services, pp. 239–252. ACM (2011)
8. Delorme, A., Perrinet, L., Thorpe, S.J.: Networks of Integrate-and-fire neurons using rank order coding b: spike timing dependant plasticity and emergence of orientation selectivity. Neurocomputing **38–40**(1–4), 539–545 (2000)
9. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: Proceedings of the 16th ACM Conference on Computer Security, CSS (2009)
10. Fedler, R., Banse, C., Krauß, Ch., Fusenig, V.: Android OS security: risks and limitations a practical evaluation, AISEC Technical Reports, AISEC-TR-2012–001 (2012)
11. Fuchs, A., Chaudhuri, A., Foster, J.: ScanDroid: automated security certification of android applications, Technical report, University of Maryland (2009)
12. Ghorbanzadeh, M., Chen, Y., Zhongmin, M., Clancy, C.T., McGwier, R.: A neural network approach to category validation of android applications. In: International Conference on

Computing, Networking and Communications, Cognitive Computing and Networking Symposium (2013)
13. Glodek, W., Harang R.R.: Permissions-based detection and analysis of mobile malware using random decision forests. In: IEEE Military Communications Conference (2013)
14. Google, UI/Application Exerciser Monkey. http://developer.android.com/tools/help/monkey.html (2013)
15. Huang, G.-B.: An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. Springer (2014). doi:10.1007/s12559-014-9255-2
16. Joseph, J.F.C., Lee, B.-S., Das, A., Seet, B,-C.: Cross-layer detection of sinking behavior in wireless ad hoc networks using ELM and FDA. IEEE IJCA **54**(14) (2012)
17. Kasabov, N.: Evolving connectionist systems: Methods and Applications in Bioinformatics, Brain study and intelligent machines. Springer Verlag, NY (2002)
18. Kohonen, T.: Self-organizing networks. In: Proceedings of the IEEE (1990)
19. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: International Conference on Knowledge Discovery and Data Mining, pp. 470–478 (2006)
20. Lange, M., Liebergeld, S., Lackorzynski, A., Peter M.: L4Android: a generic operating system framework for secure smartphones. In: ACM Workshop on SPSM (2011)
21. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability (1967)
22. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H.: Paranoid Android: versatile protection for smartphones. In: 26th Annual Computer Security Applications Conference (2010)
23. Sahs, J., Khan, L.: A Machine learning approach to android malware detection. In: European Intelligence and Security Informatics Conference (2012)
24. Scandariato, R., Walden, J.: Predicting Vulnerable Classes in an Android Application (2012)
25. Schliebs, S., Kasabov, N.: Evolving spiking neural network—a survey. Evolving Systems **4** (2), 87–98 (2013)
26. Schliebs, S., Defoin-Platel, M., Kasabov, N.: Integrated Feature and Parameter Optimization for an Evolving Spiking Neural Network, 5506, pp. 1229–1236. Springer (2009)
27. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S. J.: Data mining methods for detection of new malicious executables. In: SP '01, pp. 38. IEEE Computer Society, Washington, DC (2001)
28. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: IC Computational Intelligence and Security (2010)
29. Shabtai, A., Fledel, Y., Elovici Y.: Automated static code analysis for classifying android applications using machine learning, in CIS. In: Conference on IEEE, pp. 329–333 (2010)
30. Tesauro, G.J., Kephart, J.O., Sorkin, G.B.: Neural networks for computer virus recognition. IEEE Expert **11**(4), 5–6 (1996)
31. Thorpe, S.J., Delorme, A.: Rufin van Rullen: Spike-based strategies for rapid processing. Neural Netw. **14**(6–7), 715–725 (2001)
32. Thorpe, S.J., Gautrais, J.: Rank order coding. In: CNS '97: 6th Conference on Computational Neuroscience: Trends in Research, pp. 113–118. Plenum Press (1998)
33. www.wala.sourceforge.net/wiki/index.php
34. www.extreme-learning-machines.org/
35. www.cs.waikato.ac.nz/ml/weka
36. www.gmdhshell.com/
37. Wysoski, S.G., Benuskova, L., Kasabov, N.K.: Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. In: Advanced Concepts for Intelligent Vision Systems, pp. 1133–1142. Springer Berlin/Heidelberg (2006)