DL4MD: A Deep Learning Framework for Intelligent Malware Detection

William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye*, and Xin Li
Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506, USA

Abstract- *In the Internet-age, malware poses a serious and evolving threat to security, making the detection of malware of utmost concern. Many research efforts have been conducted on intelligent malware detection by applying data mining and machine learning techniques. Though great results have been obtained with these methods, most of them are built on shallow learning architectures, which are still somewhat unsatisfying for malware detection problems. In this paper, based on the Windows Application Programming Interface (API) calls extracted from the Portable Executable (PE) files, we study how a deep learning architecture using the stacked AutoEncoders (SAEs) model can be designed for intelligent malware detection. The SAEs model performs as a greedy layerwise training operation for unsupervised feature learning, followed by supervised parameter fine-tuning (e.g., weights and offset vectors). To the best of our knowledge, this is the first work that deep learning using the SAEs model based on Windows API calls is investigated in malware detection for real industrial application. A comprehensive experimental study on a real and large sample collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. Promising experimental results demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning methods.*

Keywords: Malware Detection, Windows Application Programming Interface (API) Calls, Deep Learning.

1 Introduction

Malware is ***mal***icious soft***ware*** disseminated to infiltrate the secrecy, integrity, and functionality of a system [8], such as viruses, worms, trojans, backdoors, spyware. With computers and the Internet being essential in everyday life, malware poses a serious threat to their security. It is estimated that one in four computers operating in the U.S. are infected with malware [24]. The threat malware poses is not only emotional, but financial as well. According to a recent report from Kaspersky Lab, up to one billion dollars was stolen in roughly two years from financial institutions worldwide, due to malware attacks [16]. As a result, the detection of malware is of major concern to both the anti-malware industry and researchers.

In order to protect legitimate users from the attacks, the majority of anti-malware software products (e.g., Comodo, Symantec, Kaspersky) use the signature-based method of detection [10], [9]. Signature is a short string of bytes, which is unique for each known malware so that its future examples can be correctly classified with a small error rate [19]. However, this method can be easily evaded by malware attackers through the techniques such as encryption, polymorphism and obfuscation [29], [2]. Furthermore, malicious files are being disseminated at a rate of thousands per day [34], making it difficult for this signature-based method to be effective. In order to combat the malware attacks, intelligent malware detection techniques need to be investigated. As a result, many researches have been conducted on intelligent malware detection by applying data mining and machine learning techniques in recent years [27], [1], [23], [32], [35]. Based on different feature representations, different kinds of classification methods, such as Artificial Neural Network (ANNs), Support Vector Machine (SVM), Naïve Bayes (NB), and Decision Tree (DT), are used for model construction to detect malware [20], [7], [27]. Most of these methods are built on shallow learning architectures. Though they had isolated success in malware detection, shallow learning architectures are still somewhat unsatisfying for malware detection problems.

Deep learning (DL), a new frontier in data mining and machine learning, is starting to be leveraged in industrial and academic research for different applications (e.g., Computer Vision) [4], [14], [22]. A multilayer deep learning architecture is of superior ability in feature learning. More importantly, deep learning architectures overcome the learning difficulty through layerwise pretraining, i.e. pretraining multiple layers of feature detectors from the lowest level to the highest level to construct the final classification model [22]. This inspires us to devise a malware detection architecture based on deep learning.

In this paper, a deep learning architecture using the stacked AutoEncoders (SAEs) model for malware detection is studied, with the input resting on Windows Application Programming Interface (API) calls extracted from the Portable Executable (PE) files. The SAEs model employs a greedy layerwise training operation for unsupervised feature learning, followed by supervised parameter fine-tuning (e.g., weights and offset vectors). To the best of our knowledge, this is the first work investigating deep learning using the SAEs model resting on Windows API calls in malware detection for real industrial application. For validation, a comprehensive experimental study on a real and large sample collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. The experimental results obtained demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning

*Corresponding author

methods and that deep learning is a promising architecture for malware detection.

The rest of the paper is structured as follows. Section 2 presents the related work. Section 3 describes the overview of our malware detection system. Section 4 introduces our proposed deep learning approach for malware detection. Section 5 evaluates the performance of our proposed method in comparison with other alternative methods in malware detection. Finally, Section 6 concludes.

2 Related work

Signature-based method is widely used in anti-malware industry [10], [9]. However, malware authors can easily evade this signature-based method through techniques such as encryption, polymorphism and obfuscation [29]. Driven by the economic benefits, the quantity, diversity and sophistication of malware has significantly increased in recent years [34]. In order to effectively and automatically detect malware from the real and large daily sample collection, new, intelligent malware detection systems have been developed by applying data mining and machine learning techniques [31], [27], [20], [33], [28], [13]. These intelligent malware detection systems are varied in their uses of feature representations and classification methods. Naïve Bayes on the extracted strings and byte sequences was applied in [27], which claimed that Naïve Bayes classifier performed better than traditional signature-based method. Kolter et al. [20] focused on static analysis of the executable files and compared Naïve Bayes, Support Vector Machine and Decision Tree based on the n-grams. Ye et al. [33] proposed IMDS performing associative classification on Windows API calls extracted from executable files. Shah et al. [28] applied various feature selection algorithms to obtain the feature sets from PE files and used Artificial Neural Networks to detect new and unknown malware. Hou et al. [13] developed the intelligent malware detection system using cluster-oriented ensemble classifiers resting on the analysis of Windows API calls. Most of these existing researches are built on shallow learning architectures.

Due to its superior ability in feature learning through multilayer deep architecture [11], deep learning is feasible to learn higher level concepts based on the local feature representations [25]. As a result, researchers have paid much attention to deep learning methods in the domains of natural language processing, computer vision, etc. [18], [6]. In recent years, limited research efforts have been devoted to the malware detection using deep learning [21], [25], [15]. Ouellette et al. [25] extracted control flow graphs to present malware samples, and used a deep probabilistic model (sum-product network) to compare the similarities between the unknown file samples and those of representative sample features from known classes of malware. Jung et al. [15] used the features of header, tags, bytecode and API calls and utilized an ensemble learner consisting of different deep learning networks (e.g., deep feed-forward neural network, deep recurrent neural network) to classify the Adobe Flash file samples. Li et al. [21] proposed a hybrid malicious code detection approach on the basis of AutoEncoder and Deep Belief Network, where AutoEncoder was used to reduce the dimensionality of data, and a Deep Belief Network was applied to detect malicious code.

Different from the previous work, based on a real and large sample collection from Comodo Cloud Security Center, resting on the analysis of Windows API calls, we attempt to explore a deep learning architecture with the SAEs model to learn generic features of malware and thus to detect newly unknown malware.

3 System architecture

Our deep learning framework for malware detection (short for DL4MD) is performed on the analysis of Windows API calls generated from the collected PE files. The system consists of two major components: feature extractor, and deep learning based classifier, as illustrated in Figure 1.

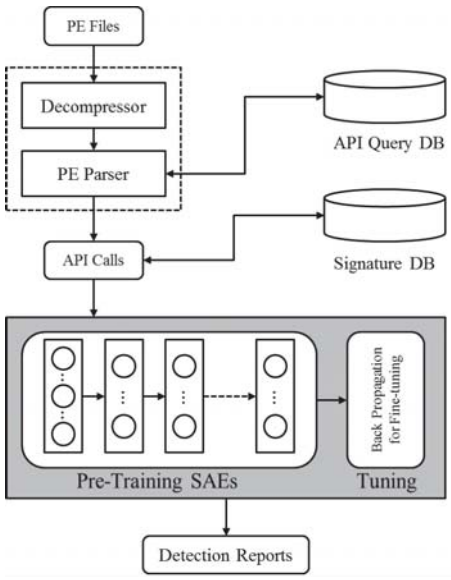

Fig. 1. DL4MD system architecture

- *Feature Extrator:* It is composed of Decompressor and PE Parser. The Decompressor needs to be employed first when a PE file is previously compressed by a binary compress tool (e.g., UPX and ASPack Shell) or embedded a homemade packer. The PE parser is used to extract the Windows API calls from each PE file. Through the API query database, the Windows API calls can be converted to a set of 32-bit global IDs representing the corresponding API functions (e.g., the API of "MAPI32.MAPIReadMail" is encoded as 0x00600F12). Then the API calls are stored as the signatures of the PE files in the signature database.
- *Deep Learning based Classifier:* Based on the Windows API calls, a deep learning architecture using SAEs model is designed to perform unsupervised feature learning, supervised fine-tuning, and thus malware detection. (See Section 4 for details.)

TABLE I
AN EXAMPLE DATASET OF SAMPLE FILES

File	Extracted API Calls	Feature Vector	Class
f_1	API_1, API_3	$< 1,0,1,0,0 >$	c_m
f_2	API_2, API_3, API_5	$< 0,1,1,0,1 >$	c_b
f_3	API_2, API_3, API_4, API_5	$< 0,1,1,1,1 >$	c_b
f_4	API_3	$< 0,0,1,0,0 >$	c_m
f_5	API_1, API_2, API_4	$< 1,1,0,1,0 >$	c_g

4 Proposed method

4.1 Problem definition

Resting on the analysis of Windows API calls, which can reflect the behavior of program code pieces [32] (e.g., the API "*GetModuleFileNameA*" in "*Kernel32.DLL*" can be used to retrieve the complete path of the file that contains the specified module of current process), a file f_i in our dataset D is denoted to be of the form $\langle A_{f_i}, C_{f_i} \rangle$, where A_{f_i} is the Windows API call feature vector of file f_i, and C_{f_i} is the class label of file f_i (where $C_{f_i} \in \{c_m, c_b, c_g\}$; c_m denotes malicious, c_b benign, and c_g unknown). Let d be the number of all extracted Windows API calls in the dataset D. The feature of each file can be represented by a binary feature vector:

$$A_{f_i} =< a_{i1}, a_{i2}, a_{i3}, ..., a_{id} >, \quad (1)$$

where $a_{ij} = \{0,1\}$ (i.e., if f_i contains API_j, $a_{ij} = 1$; otherwise, $a_{ij} = 0$). Table I shows a sample example.

Our malware detection problem statement can be stated as follows: *Given a dataset D as defined above, assign a label (i.e., c_m or c_b) to each unlabeled file (i.e., c_g) based on its feature vector*.

Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning models and can thus achieve comparable or better performance [11]. Typical deep learning models include stacked AutoEncoders (SAEs), Deep Belief Networks with Restricted Boltzmann Machine, Convolutional Neural Networks [5], etc. In this paper, we explore a deep learning architecture with SAEs model for malware detection. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network.

4.2 AutoEncoder

An AutoEncoder, also called AutoAssociator, is an artificial neural network used for learning efficient codings [30]. Architecturally, the form of an AutoEncoder is with an input layer, an output layer and one or more hidden layers connecting them. The goal of an AutoEncoder is to encode a representation of the input layer into the hidden layer, which is then decoded into the output layer, yielding the same (or as close as possible) value as the input layer [4]. In this way, the hidden layer acts as another representation of the feature space, and in the case when the hidden layer is narrower (has fewer nodes) than the input/output layers, the activations of the final hidden layer can be regarded as a compressed representation of the input [4], [30], [5]. Figure 2 illustrates a one-layer AutoEncoder model with one input layer, one hidden layer, and one output layer.

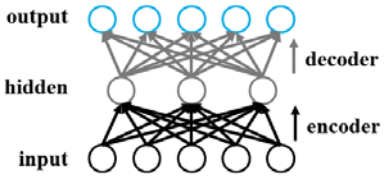

Fig. 2. A one-layer AutoEncoder model

Given a training set $\mathbf{X}$ of n samples $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n\}$, where $\mathbf{x}_i$ ($i \in \{1, ..., n\}$) is a d_0-dimensional feature vector, i.e., $\mathbf{x}_i \in R^{d_0}$, and a sigmoid function denoted as

$$s(\mathbf{t}) = \frac{1}{1 + \exp^{-\mathbf{t}}}, \quad (2)$$

the AutoEncoder framework can be rigorously defined as follows [30], [22].

Encoder: In order to transform an input vector $\mathbf{x_i}$ into a hidden representation vector $\mathbf{y_i}$, the **encoder**, a deterministic mapping f_θ, is utilized. Its typical form is an affine mapping followed by a nonlinearity [30]:

$$\mathbf{y_i} = f_\theta(\mathbf{x_i}) = s(\mathbf{W}\mathbf{x_i} + \mathbf{b}), \quad (3)$$

where $\mathbf{W}$ is a $d_0 \times d_1$ weight matrix, d_1 is the number of hidden units, $\mathbf{b}$ is an offset vector of dimensionality d_1, and θ is the mapping parameter set $\theta = \{\mathbf{W}, \mathbf{b}\}$.

Decoder: The resulting hidden representation $\mathbf{y_i}$ is then mapped back to a reconstructed d_0-dimensional vector $\mathbf{z_i}$ in the input space, using the **decoder** $g_{\theta'}$. Its typical form is again an affine mapping optionally followed by a nonlinearity [30], i.e., either $\mathbf{z_i} = g_{\theta'}(\mathbf{x_i}) = \mathbf{W}'\mathbf{y_i} + \mathbf{b}'$ or

$$\mathbf{z_i} = g_{\theta'}(\mathbf{x_i}) = s(\mathbf{W}'\mathbf{y_i} + \mathbf{b}'), \quad (4)$$

where $\mathbf{W}'$ is a $d_1 \times d_0$ weight matrix, $\mathbf{b}'$ is also an offset vector of dimensionality d_0, and $\theta' = \{\mathbf{W}', \mathbf{b}'\}$.

Typically, the number of hidden units is much less than number of visible (input/output) ones ($d_1 < d_0$). As a result, when passing data through such a network, it first compresses (encodes) input vector to "fit" in a smaller representation, and then tries to reconstruct (decode) it back. The task of training is to minimize an error or reconstruction (using Equation 5), i.e. find the most efficient compact representation (encoding) for input data (Equation 6).

$$E(\mathbf{x}, \mathbf{z}) = \frac{1}{2} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{z}_i||^2. \quad (5)$$

$$\theta = \{\mathbf{W}, \mathbf{b}\} = \arg\min_{\theta} E(\mathbf{x}, \mathbf{z}). \quad (6)$$

Algorithm 1 illustrates the training procedure of AutoEncoder in our application. Note that other parameterized functions and loss functions can also be used for the encoder or decoder in the AutoEncoder framework [30].

Input: Data set $\mathbf{X}$ with n training samples: $\mathbf{x}_i =< A_{f_i}, C_{f_i} >$, where $i \in \{1, ..., n\}$ and $C_{f_i} \in \{c_m, c_b\}$
Output: Parameter set $\theta = \{\mathbf{W}, \mathbf{b}\}$

Initialize ($\mathbf{W}$, $\mathbf{b}$);
while *training error E hasn't converged or the designated iteration hasn't reached* **do**
  for *each input* $\mathbf{x}_i$ **do**
    Compute activations $\mathbf{y}_i$ at the hidden layer, and obtain an output $\mathbf{z}_i$ at the output layer;
  end
  Calculate the training error $E(\mathbf{x}, \mathbf{z})$;
  Backpropagate the error through the net and update parameter set $\theta = \{\mathbf{W}, \mathbf{b}\}$;
end

Algorithm 1: The algorithm for training AutoEncoder in malware detection

4.3 Deep learning architecture with SAEs

To form a deep network, an SAE model is created by daisy chaining AutoEncoders together, known as stacking: the output of one AutoEncoder in the current layer is used as the input of the AutoEncoder in the next [5]. More rigorously, with an SAE deep network with h layers, the first layer takes the input from the training dataset and is trained simply as an AutoEncoder. Then, after the k^{th} hidden layer is obtained, its output is used as the input of the $(k+1)^{th}$ hidden layer, which is trained similarly. Finally, the h^{th} layer's output is used as the output of the entire SAE model. In this manner, AutoEncoders can form a hierarchical stack. Figure 3 illustrates a SAEs model with h hidden layers.

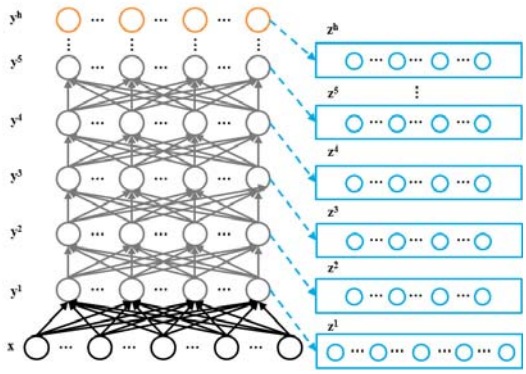

Fig. 3. A stacked AutoEncoders model

To use the SAEs for malware detection, a classifier needs to be added on the top layer. In our application, the SAEs and the classifier comprise the entire deep architecture model for malware detection, which is illustrated in Figure 4.

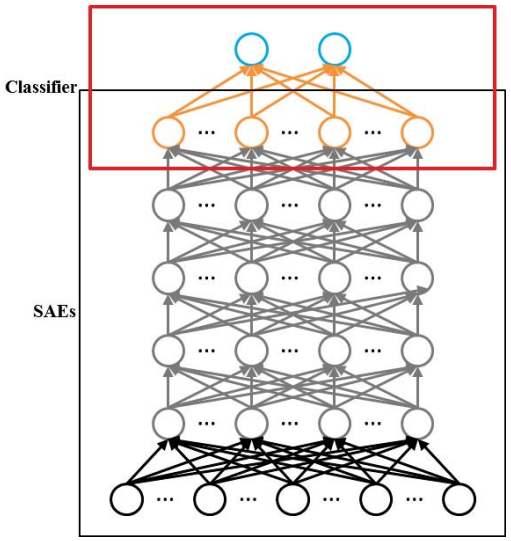

Fig. 4. Deep learning model for malware detection.

It is simple to train the deep network using Back Propagation (BP) with the gradient-based optimization technique, however, deep networks trained in this way are known to have poor performance. Fortunately, the greedy layerwise unsupervised learning algorithm developed by Hinton et al. [12] has overcome this problem. The key point of this algorithm is to pretrain the deep network layer by layer in a bottom-up manner and then fine-tune the parameters by applying BP in a top-down manner, which obtains better results. Resting on [12], [5], the training algorithm for malware detection using a deep learning architecture with SAEs is described in Algorithm 2.

Input: Data set $\mathbf{X}$ with n training samples: $\mathbf{x}_i =< A_{f_i}, C_{f_i} >$, where $i \in \{1, ..., n\}$ and $C_{f_i} \in \{c_m, c_b\}$; h: number of hidden layer; k_j: number of neurons for each layer, where $j \in \{1, ..., h\}$
Output: Parameter sets θs

for *each layer* $l (l \in \{1, ..., h\})$ *in SAEs* **do**
  Use Algorithm 1 to train the AutoEncoder at each layer;
end
Initialize ($\mathbf{W}^{h+1}$, $\mathbf{b}^{h+1}$) at the classifier layer;
Calculate the labels for each training sample $\mathbf{x}_i$;
Perform BP in a supervised way to tune the parameter sets θs of all layers;

Algorithm 2: The algorithm for training the deep learning network with SAEs in malware detection

5 Experimental results and analysis

In this section, we conduct two sets of experiments based on a real and large sample collection obtained from Comodo Cloud Security Center to fully evaluate the performance of our deep learning model in malware detection: (1) In the first

TABLE II
PERFORMANCE MEASURES IN MALWARE DETECTION

Measure	Description
TP	Number of files correctly classified as malicious
TN	Number of files correctly classified as benign
FP	Number of files mistakenly classified as malicious
FN	Number of files mistakenly classified as benign
TP Rate (TPR)	$\frac{TP}{TP+FN}$
FP Rate (FPR)	$\frac{FP}{FP+TN}$
Accuracy (ACY)	$\frac{TP+TN}{TP+TN+FP+FN}$

Remark: TP: True Positive, TN: True Negative, FP: False Positive, and FN: False Negative.

set of experiments, we evaluate the deep learning networks with different parameters (i.e., different numbers of hidden layers and neurons); (2) In the second set of experiments, we conduct a comparison between our proposed method and other shallow learning based classification methods (i.e., ANN, SVM, NB, and DT).

5.1 Experimental setup

The dataset obtained from Comodo Cloud Security Center contains $50,000$ file samples, where $22,500$ are malware, $22,500$ are benign files, and $5,000$ are unknown (with the analysis by the anti-malware experts of Comodo Security Lab, $2,500$ of them are labeled as malware and $2,500$ of them are benign). In our experiments, those $45,000$ file samples are used for training, while the $5,000$ unknown files are used for testing. $9,649$ Windows API calls are extracted from these $50,000$ file samples, so all the file samples can be represented as binary feature vectors with $9,649$-dimensions (described in Section 4.1). To quantitatively validate the experimental results, we use the performance measures shown in Table II. All experiments are conducted in the environment: 64 Bit Windows 8.1 on an Intel (R) Core (TM) i7-4790 Processor (3.60GHz) with 16GB of RAM, using MySQL and C++.

5.2 Evaluations of different deep networks

In this section, based on the dataset described in Section 5.1, we evaluate the deep learning networks with different parameters (i.e., different numbers of hidden layers and neurons). The results in Table III and Figure 5 demonstrate the effectiveness of our proposed algorithm in malware detection. The deep learning model with 3 hidden layers and 100 neurons at each hidden layer is superior to other deep networks in training and testing accuracy for malware detection.

5.3 Comparisons between deep learning and other shallow learning based classification methods

In this section, using the same dataset described in Section 5.1, we conduct a comparison between our proposed

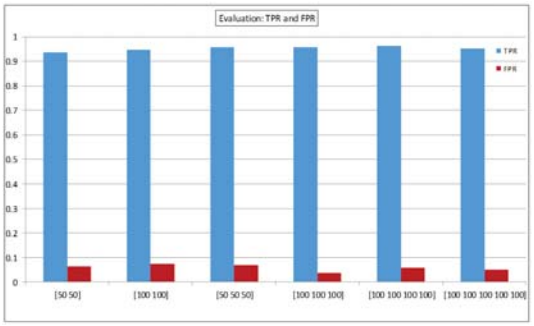

Fig. 5. Comparisons of testing TPR and FPR between different deep networks

deep learning framework (DL4MD) and other shallow learning based classification methods (i.e., Artificial Neural Network (ANN), Support Vector Machine (SVM), Naïve Bayes (NB), and Decision Tree (DT)) in malware detection. The results in Table IV, Figure 6 and Figure 7 show that our proposed deep learning framework (DL4MD) outperform ANN, SVM, NB, and DT in malware detection.

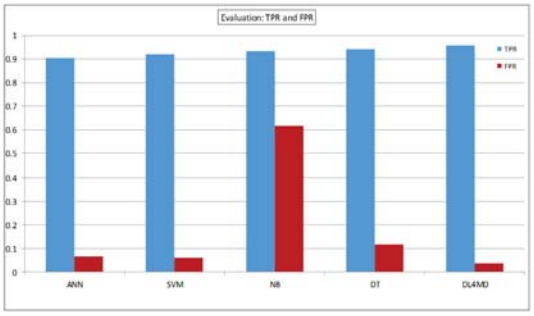

Fig. 6. Comparisons of testing TPR and FPR between ANN, SVM, NB, DT, and DL4MD

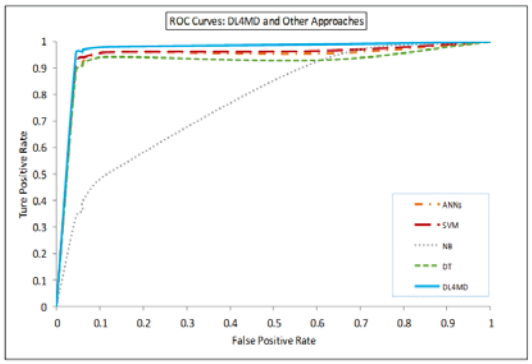

Fig. 7. Comparisons of ROC curves between ANN, SVM, NB, DT, and DL4MD

Our proposed deep learning framework (DL4MD) also performs well in detection efficiency: it just takes about 0.1 second for each unknown sample prediction, including feature extraction. This makes our system a practical solution for intelligent malware detection in real industrial application.

TABLE III
EVALUATION OF DIFFERENT DEEP NETWORKS

Training

Hidden Layers	Neurons	TP	FP	TN	FN	ACY
2	[50 50]	21,859	1,434	21,066	641	0.9539
2	[100 100]	22,142	1,460	21,040	358	0.9596
3	[50 50 50]	22,110	1,295	21,205	390	0.9626
3	**[100 100 100]**	**22,035**	**953**	**21,547**	**465**	**0.9685**
4	[100 100 100 100]	22,150	1,178	21,322	350	0.9660
5	[100 100 100 100 100]	22,055	977	21,523	445	0.9684

Testing

Hidden Layers	Neurons	TP	FP	TN	FN	ACY
2	[50 50]	2,368	161	2,339	132	0.9414
2	[100 100]	2,391	185	2,315	109	0.9412
3	[50 50 50]	2,396	170	2,330	104	0.9452
3	**[100 100 100]**	**2,396**	**114**	**2,386**	**104**	**0.9564**
4	[100 100 100 100]	2,408	147	2,353	92	0.9550
5	[100 100 100 100 100]	2,384	128	2,372	116	0.9512

TABLE IV
COMPARISONS BETWEEN DEEP LEARNING AND OTHER SHALLOW LEARNING BASED CLASSIFICATION METHODS

Training

Method	TP	FP	TN	FN	ACY
ANN	21,338	1,781	20,719	1,162	0.9346
SVM	21,610	1,576	20,924	890	0.9452
NB	21,532	9,940	12,560	968	0.7576
DT	21,630	1,560	20,940	870	0.9460
DL4MD	**22,035**	**953**	**21,547**	**465**	**0.9685**

Testing

Method	TP	FP	TN	FN	ACY
ANN	2,264	163	2,337	236	0.9202
SVM	2,305	152	2,348	195	0.9306
NB	2,332	1,541	959	168	0.6582
DT	2,357	292	2,208	143	0.9130
DL4MD	**2,396**	**114**	**2,386**	**104**	**0.9564**

6 Conclusion and future work

In this paper, based on Windows API calls extracted from a real and large file sample collection, we design a deep learning framework using the SAEs model for malware detection, which consists of two phases: *unsupervised pre-training* and *supervised backpropagation*. The SAEs model performs as unsupervised pretraining in a greedy layer-wise fashion putting the parameters of all layers in a region of parameter space in a bottom-up way, and then supervised BP is adopted to tune the multilayer model's parameters in a top-down direction. To the best of our knowledge, this is the first work investigating deep learning using the SAEs model based on Windows API calls in malware detection for real industrial application. A comprehensive experimental study on a real and large file collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. The experimental results obtained demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning methods and that deep learning is a viable architecture for malware detection.

In our future work, we will further explore how sparsity constraints are imposed on AutoEncoder and how sparse SAEs can be designed to further improve malware detection effectiveness. Meanwhile, it would be interesting to investigate other deep learning models for malware detection.

Acknowledgment

The authors would also like to thank the anti-malware experts of Comodo Security Lab for the data collection as well as helpful discussions and supports. This work is partially supported by the U.S. National Science Foundation under grant CNS-1618629.

References

[1] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Ahanian, and J. Nazario. Automated classification and analysis of internet malware. In *RAID 2007, LNCS*, 178-197, 2007.

[2] P. Beaucamps, and E. Filiol. On the possibility of practically obfuscating programs towards a unified perspective of code protection. In *Journal in Computer Virology*, 3 (1), 2007.

[3] Y. Bengio, and Y. LeCun. Scaling Learning Algorithms towards AI. In *Large-scale kernel machines*, 34(5), 2007.

[4] Y. Bengio. Learning Deep Architectures for AI. In *Foundations and Trends in Machine Learning*, Vol 2(1), 1-127, 2009.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy Layer-Wise Training of Deep Networks. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*, 153-160, 2007.

[6] R. Collobert, and J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of 25th ICML*, 160-167, 2008.

[7] R. A. Dunne. A Statistical Approach to Neural Networks for Pattern Recognition. In *Wiley-Interscience, 1st edition*, 2007.

[8] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A Survey on Automated Dynamic Malware Analysis Techniques and Tools. In *ACM CSUR*, Vol 44(2), 6:1-6:42, 2008.

[9] E. Filiol. Malware pattern scanning schemes secure against blackbox analysis. In *J. Comput. Virol*, Vol 2(1), 35-50, 2006.

[10] E. Filiol, G. Jacob, and M. L. Liard. Evaluation methodology and theoretical model for antiviral behavioural detection strategies. In *J. Comput. Virol*, Vol 3(1), 27-37, 2007.

[11] G. E. Hinton, and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. In *Science*, Vol 313(5786), 504-507, 2006.

[12] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. In *Neural Computation*, Vol 18, 1527-1554, 2006.

[13] S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye. Cluster-Oriented Ensemble Classifiers for Malware Detection. In *IEEE International Conference on Sematic Computing (IEEE ICSC)*, 189-196, 2015.

[14] W. Huang, G. Song, H. Hong, and K. Xie. Deep Architecture for Traffic Flow Prediction: Deep Belief Networks With Multitask Learning. In *IEEE Transactions on Intelligent Transportation Systems*, Vol 15(5), 2191-2201, 2014.

[15] W. Jung, S. Kim, and S. Choi. Poster: Deep Learning for Zero-day Flash Malware Detection. In *36th IEEE Symposium on Security and Privacy*, 2015.

[16] Kaspersky Lab. The Great Bank Robbery. In *http://www.kaspersky.com/about/news/virus/2015/Carbanak-cybergang-steals-1-bn-USD-from-100-financial-institutions-worldwide*, 2015.

[17] Kaspersky Lab. Kaspersky Anti-Virus SDK v.8 Core Detection Technologies. In *White Paper*, August, 2009.

[18] K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning Convolutional Feature Hierachies for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS 2010)*, 23, 2010.

[19] J. Kephart, and W. Arnold. Automatic extraction of computer virus signatures. In *Proceedings of 4th Virus Bulletin International Conference*, 178-184, 1994.

[20] J. Kolter, and M. Maloof. Learning to detect malicious executables in the wild. In *SIGKDD*, 2004.

[21] Y. Li, R. Ma, and R. Jiao. A Hybrid Malicious Code Detection Method based on Deep Learning. In *International Journal of Security and Its Applications*, Vol 9(5), 205-216, 2015.

[22] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Wang. Traffic Flow Prediction With Big Data: A Deep Learning Approach. In *IEEE Transactions on Intelligent Transportation Systems*, Vol 16(2), 865-873, 2015.

[23] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Cloud-Based Malware Detection for Evolving Data Streams. In *ACM TMIS*, Vol 2(3), 16:1-16:27, 2008.

[24] Organisation for Economic Co-operation and Development. Malicious software (malware):Asecurity threat to the internet economy. White Paper, June, 2008.

[25] J. Ouellette, A. Pfeffer, and A. Lakhotia. Countering Malware Evolution Using Cloud-Based Learning. In *8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*, 85-94, 2013.

[26] Y. Park, Q. Zhang, D. Reeves, and V. Mulukutla. AntiBot: Clustering Common Semantic Patterns for Bot Detection. In *IEEE 34th Annual Computer Software and Applications Conference*, 262-272, 2010.

[27] M. Schultz, E. Eskin, and E. Zadok. Data mining methods for detection of new malicious executables. In *Proccedings of IEEE Symposium on Security and Privacy*, 2001.

[28] S. Shah, H. Jani, S. Shetty, and K. Bhowmick. Virus Detection using Artificial Neural Networks. In *International Journal of Computer Applications*, vol. 84(5), 2013.

[29] A. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables (save). In *Proceedings of the 20th ACSAC*, 326-334, 2004.

[30] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. In *Journal of Machine Learning Research*, Vol 11, 3371-3408, 2010.

[31] J. Wang, P. Deng, Y. Fan, L. Jaw, and Y. Liu. Virus detection using data mining techniques. In *Proccedings of ICDM03*, 2003.

[32] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang. An intelligent PE-malware detection system based on association mining. In *Journal in Computer Virology*, Vol 4, 323-334, 2008.

[33] Y. Ye, D. Wang, T. Li, and D. Ye. IMDS: Intelligent Malware Detection System. In *Proceedings of the 13th ACM SIGKDD*, 1043-1047, 2007.

[34] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu. Combining File Content and File Relations for Cloud Based Malware Detection. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD)*, 222-230, 2011.

[35] Y. Fan, Y. Ye and L. Chen. Malicious sequential pattern mining for automatic malware detection. In *Expert Systems with Applications*, Vol 52, 16-25, 2016.