International Journal of Security and Its Applications
Vol. 9, No. 5 (2015), pp. 205-216
http://dx.doi.org/10.14257/ijsia.2015.9.5.21

A Hybrid Malicious Code Detection Method based on Deep Learning

Yuancheng Li, Rong Ma and Runhai Jiao

School of Control and Computer Engineering,
North China Electric Power University, Beijing, China
ycli@ncepu.edu.cn, acelin007@163.com, runhaijiao@ncepu.edu.cn

Abstract

In this paper, we propose a hybrid malicious code detection scheme based on AutoEncoder and DBN (Deep Belief Networks). Firstly, we use the AutoEncoder deep learning method to reduce the dimensionality of data. This could convert complicated high-dimensional data into low dimensional codes with the nonlinear mapping, thereby reducing the dimensionality of data, extracting the main features of the data; then using DBN learning method to detect malicious code. DBN is composed of multilayer Restricted Boltzmann Machines (RBM, Restricted Boltzmann Machine) and a layer of BP neural network. Based on unsupervised training of every layer of RBM, we make the output vector of the last layer of RBM as the input vectors of BP neural network, then conduct supervised training to the BP neural network, finally achieve the optimal hybrid model by fine-tuning the entire network. After inputting testing samples into the hybrid model, the experimental results show that the detection accuracy getting by the hybrid detection method proposed in this paper is higher than that of single DBN. The proposed method reduces the time complexity and has better detection performance.

Keywords*: Malicious code Detection, AutoEncoder, DBN, RBM, deep learning*

1. Introduction

Malicious code is the software which intentionally damage or destroy the function of system through adding, changing, deleting some code by unauthorized users in normal circumstances. In recent years, malicious code causing far-reaching influence mainly includes: viruses, Worm, Trojan horse, etc. According to the statistical results in [1], in 2010, Symantec recorded more than 3,000,000,000 malicious code attacks, and monitoring more than 280,000,000 independent variant malicious code samples. Compared to 2009, there is growth of 93% for the attack based on the Web. With the increase in the number of malicious code, this shows that the harm and loss is growing. As an important technology of network security, intrusion detection discovers and recognizes intrusion behaviors or attempts in the system through the collection and analysis of key data in the network and computer system. Efficient, accurate identification of malicious code can improve the efficiency of intrusion detection, therefore, malicious code analysis and detection is a key problem in intrusion detection technology.

For detection of malicious code, according to the detected position it is currently divided into two approaches host-based and network-based [2]: Network-based detection methods, including Honeypot-based approach [3-4], and based on Deep packet Inspection [5]; Host-based detection methods, including check sum-based approach [6], signature-based approach [7-9], heuristic data mining approach [10]. The data mining method adopted many machine learning methods, which had an effective detection of unknown malicious code through learning the characteristics of malicious code and the normal code [11] reviewed a variety of feature-extraction methods and machine learning

ISSN: 1738-9984 IJSIA

methods in a variety of malicious code detection applications, including naive Bayes, decision trees, artificial neural networks, Support Vector Machine, *etc*., [12] proposed a static system call sequences based on N-gram and two automatic feature-selection methods, and adopted K-nearest neighbor algorithm, SVM, decision tree as the classifier. The literature [13] presented a malicious code behavior feature extraction and detection method based on semantics to obtain the behavior of malicious code which has great anti-jamming capabilities.

Although the above methods have achieved certain results in the aspect of malicious code detection, there are still some problems. Such as, feature-extraction is not appropriate, the detection rate and the detection accuracy are not high, and the complexity of the algorithm is high. This paper selects KDDCUP'99 data set as experimental data, and proposes a hybrid malicious code detection model based on deep learning; Based on the AutoEncoder for data dimensionality reduction, this paper proposes to set DBN as a classifier. For the malicious code behavior, using multiple deep learning achieved better effects than surface learning model. Finally, this method improves the malicious code detection rate and detection accuracy, and reduces the time complexity of the hybrid model.

2. Hybrid Malicious Code Detection Model based on Deep Learning

Network data usually contains the normal data and the malicious data. Malicious code detection is to differentiate between the normal data and malicious code data separately, so essentially it belongs to binary classification problems. To get a good performance of the malicious code detection model, there are two aspects of work need to be done: Firstly, finding the essential characteristics of malicious code data; secondly, constructing a good performance of classifier model to accurately differentiate the malicious data from the normal data. In this paper, we make use of the advantages of deep learning, the organic integration of two deep learning methods, AutoEncoder and DBN. This hybrid model extracts the essence of malicious code data, reduces the complexity of the model, and improves the detection accuracy of malicious code.

2. 1 AutoEncoder Dimensionality Reduction

AutoEncoder [14] is a kind of deep learning method for learning efficient code which is proposed by G. E. Hinton in 2006. Through the study of the compression coding of specified set of data, it can achieve the purpose of data dimensionality reduction. AutoEncoder structure is divided into part of encoder and decoder, including input layer, hidden layer, output layer. The cross section between encoder and decoder named code layer is the core of AutoEncoder that can reflect the essential characteristics of high dimensional data set with nested structure, and to set the intrinsic dimensions of high-dimensional data sets. When the number of hidden layer neurons are less than the number of input layer and output layer neurons, we can get the compressed vector of input layer called the data dimensionality reduction.

AutoEncoder consists of three steps, which are pretraining, Unrolling and fine-tuning process [14], as shown in Figure 1.

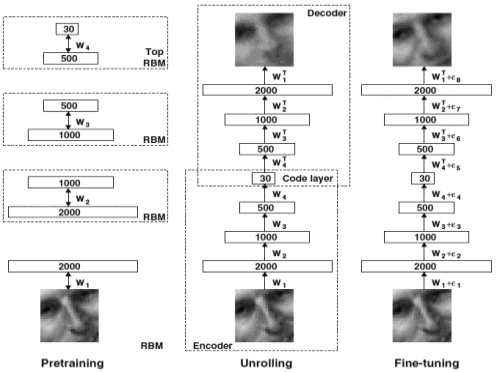

Figure 1. AutoEncoder Structure

In the pretraining process, we set the output of each RBM hidden layer neuron as the input of the next RBM. RBM consists of the visible units and hidden units. We use the vector V and H represent the visible units and the hidden unit state respectively. The structure is shown in Figure 2.

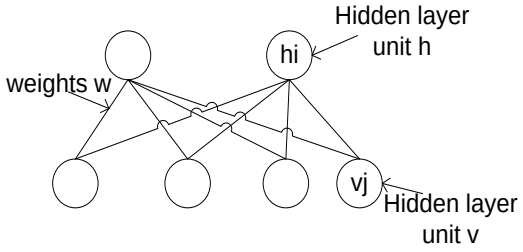

Figure 2. The Network Structure of RBM

Where v_i denotes the state of the i visible unit, h_j denotes the state of the j hidden unit, visible and hidden units meet the energy formula (1):

$$E(v,h) = -\sum_{i \in V} b_i v_i - \sum_{j \in H} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{1}$$

In the process of adjustment of weight training, firstly we update the state of the hidden layer neuron, and then update the state of the visible layer, thus get the adjusting weights. The weight updating rule as shown in formula (2):

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} = w_{ij}(t) + \varepsilon\left(\langle v_i h_j \rangle^{+} - \langle v_i h_j \rangle^{-}\right) \tag{2}$$

Where Δw_{ij} denotes the weight adjustment, $w_{ij}(t)$ denotes the connection weights(when in step t between the i, j neuron), ε denotes the learning rate, $\langle v_i h_j \rangle^{+}$ denotes the average forward correlation (Equal to the output of product of neurons in the hidden and visible neurons), $\langle v_i h_j \rangle^{-}$ denotes the average reverse correlation.

After the pre-training is completed, combining the current RBM output unit with the next RBM input unit as the independent layer. Unrolling process is to connect these independent RBM into a multi-layered AutoEncoder, the Unrolling process as shown in Figure 3.

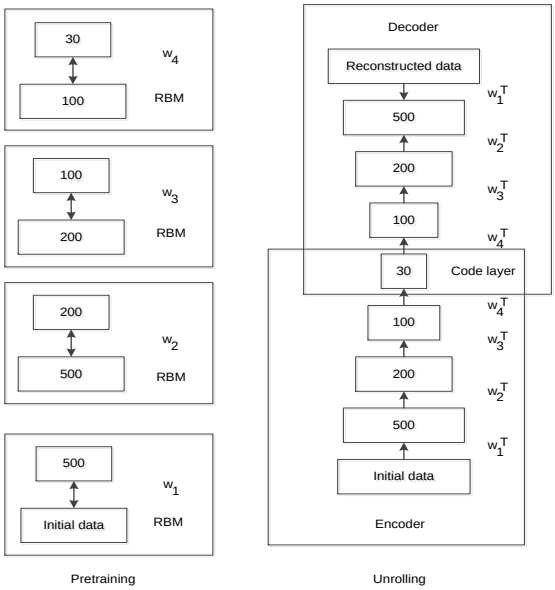

Figure 3. The Unrolling Process of AutoEncoder

Fine-tuning process is the process that does the further adjustments to the initial weights after pretraining process to get optimal weights. We mainly use the multiclass cross-entropy error function [15] for evaluation.

The multiclass cross-entropy error function is the difference between the measurement of target probability distribution and the actual probability distribution, that the smaller, the two distributions are similar, and the better. AutoEncoder uses BP algorithm to adjust the weights of the multiclass cross-entropy error function, as shown in formula (3):

$$H = [-\sum_i y_i \log \hat{y}_i - \sum_i (1-y_i)\log(1-\hat{y}_i)] \tag{3}$$

Where y_i denotes the characteristics of the data sample values, $\hat{y}_i$ denotes the Characteristics of the data sample after reconstruction.

AutoEncoder adjusts the weights in the fine-tuning process, out layer weight adjustment rules shown as formula (4):

$$\Delta w_{ij} = -\alpha \frac{\partial H_m}{\partial w_{ij}} = \alpha (t_i - y_i) O_j \tag{4}$$

Hidden layer weights adjustment rules shown as formula (5):

$$\begin{aligned} \Delta w_{ij} &= -\alpha \frac{\partial H_m}{\partial w_{ij}} = -\alpha \frac{\partial H_m}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} = -\alpha \frac{\partial H_m}{\partial net_i} O_j \\ &= -\alpha \frac{\partial H_m}{\partial O_i} \frac{\partial O_i}{\partial net_i} O_j = -\alpha \frac{\partial H_m}{\partial O_i} O_j (1-O_j) O_j \end{aligned} \tag{5}$$

Where α denotes the adjustment step, O_j denotes the upper output neurons.

2. 2 DBN Deep Learning Structure

DBN is a deep learning machine which consists of an unsupervised multi-layer RBM network and a supervised BP network. Each layer unit captures highly relevant implicit correlations from the hidden units of the front layer. The adjacent layers of the DBN can be decomposed into a single limited RBM, shown as Figure 4. In Figure 4, deep belief networks shown as Figure (1), and Figure (2) indicated that the use of each low layer RBM as input data for the training of the next RBM, get a set of RBM by the greedy learning.

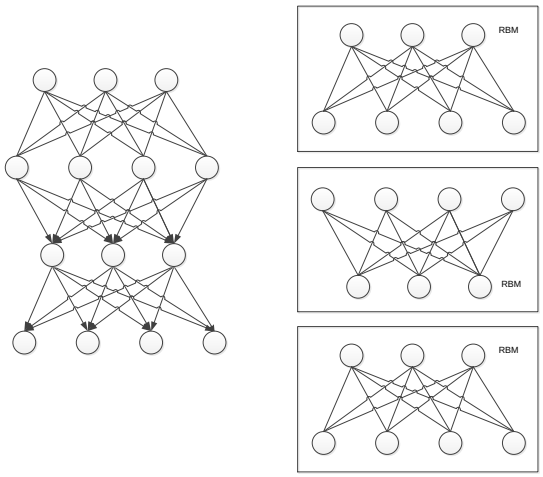

Figure 4. The RBM Structure and the Corresponding DBN Network

DBN training process is divided into two steps: The first step, train each layer of RBM separately by the unsupervised way; The second step, BP neural network in the last layer of DBN, we set the output vector of the last RBM as the input vector of BP neural network, then do the supervised training to entity relation classifier.

The paper [15] believes that, in the typical DBN which has one hidden layer the relationship between visual layer v and hidden layer h can be expressed as formula (6):

$$P\left(v,h^1,\cdots,h^l\right)=\left(\prod_{k=1}^{l-2}P\left(h^k\mid h^{k+2}\right)\right)P\left(h^{l-2},h^l\right) \tag{6}$$

As Figure 2 shown, RBM are mutually connected by the visible and the hidden layers. The connection matrix and the biases between the layers are get by unsupervised greedy algorithm. In specific training process, firstly, mapping the visual unit v_i to the hidden layer unit h_j; then, reversely reconstructing the v_i using h_j; Repeating this process, and updating the values of the connection matrix and the biases unless the reconstruction error is acceptable. Associated difference between hidden layer units and visual layer units will form the basis for each weight update. Mapping probability of hidden layer units and visual layer units shown as formula (7) and (8):

$$p\left(h_j=1\mid v;\theta\right)=\sigma\left(\sum_{i=1}^{I}w_{ij}v_i+a_j\right) \tag{7}$$

$$p\left(v_i=1\mid h;\theta\right)=\sigma\left(\sum_{j=1}^{I}w_{ij}h_i+b_j\right) \tag{8}$$

Where w_{ij} denotes the connection weights between the visual layer units and hidden layer units, b_i and a_j denotes biases respectively, sigmoid function denotes the incentive function. By using the gradient of the log likelihood probability $\log p\left(v,h;\theta\right)$, we derive the RBM weight update rule, as shown in formula (9):

$$\Delta w_{ij}=\left\langle E_{data}\left(v_ih_j\right)\right\rangle-\left\langle E_{\mathrm{mod}el}\left(v_ih_j\right)\right\rangle \tag{9}$$

Where $\langle\ \rangle$ denotes the expectation value, $\left\langle E_{data}\left(v_ih_j\right)\right\rangle$ denotes the expectation value defined in the model. Because $\left\langle E_{\mathrm{mod}el}\left(v_ih_j\right)\right\rangle$ is difficult to calculate, we always use the Gibbs sampling replace $\left\langle E_{\mathrm{mod}el}\left(v_ih_j\right)\right\rangle$ by using the contrast gradient divergence algorithm which is similar to the gradient. Through a combination of bottom-up RBMs which have carried out massive learnings layer by layer can construct an initial DBN.

Then fine tune the whole DBN from the back to the front by the supervised learning method which is similar to the traditional BP neural network. Finally, we can establish the trained DBN model.

2. 3 Hybrid Malicious Code Detection based on DBN and AutoEncoder

Deep learning has nonlinear mapping of the deep structure with the multilayer which has the benefits complex function can be expressed with fewer parameters. Compared with surface learning, it can realize complex function approximation, and has strong ability for the massed learning of the essential characteristics of data set from a few samples. Based on the above considerations, this paper proposes a hybrid malicious code detection model based on deep learning; Reducing dimensionality of the data by using the AutoEncoder's space mapping ability of different dimensionality, then abstracting the main characteristics. Based on this, setting DBN as the classifier for several times deep learnings. Then improving the detection accuracy, and reducing the time complexity of the hybrid model. Figure 5 depicts the process of mixing pre-trained detection algorithm.

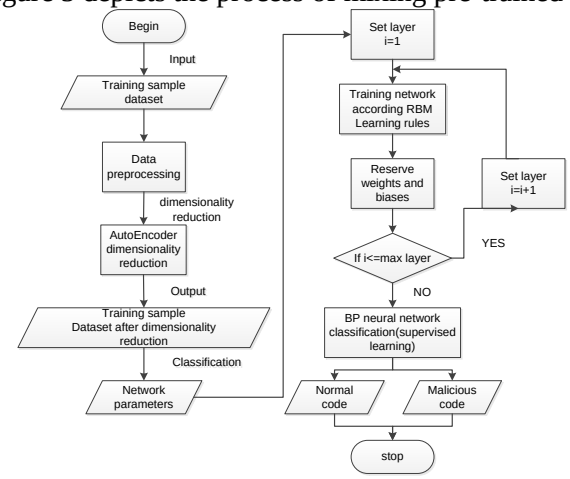

Figure 5. A DBN Malicious Code Detection Method based on AutoEncoder Dimensionality Reduction

The hybrid detection algorithm is described as follows:

(1) Initialization, input training samples; then digitizing and normalizing the input data;

(2) Reducing the dimension, AutoEncoder was used to realize the feature mapping;

(3) Input eigenvector with dimensionality reduction, network parameter to initialize DBN classifier;

(4) Set the layer i=1;

(5) Train the network layer by layer according to RBM learning rules, then save the result including the weights and biases;

(6) If i<=max layer, set i=i+1; when i>max layer, do the supervised learning for BP network;

(7) Input the test samples into the trained classifier to detect malicious code and the normal code.

3. Experimental Results and Analysis

3.1 Analysis and Pretreatment of Experimental Data

In this paper, KDDCUP'99 dataset [16] was used to detect malicious code data. They include five categories: probe, UZR (User to Root), RZL (Remote to Local), DoS (Denial-of-Service) as well as Normal data. This paper adopted 10% of the samples of KDDCUP'99 as a dataset, containing a total of 494,021 training data and 311,029 testing

data. In the dataset of KDDCUP'99, each data contains 41 properties. There are two types of data: numerical and character type. For numerical data, we can treat it directly as number; for the character of character data, we can achieve numeric in the standard method of keywords. To eliminate the effects caused by differences of the magnitude, and to reduce the excessive reliance on individual characteristics in the process of classification, we need to normalize data.

Firstly, each feature was standardized according to the formula (10)

$$x'_{ij} = \frac{x_{ij} - AVERAGE_j}{STAND_j} \tag{10}$$

$$AVERAGE_j = \frac{1}{n}\left(x_{1j} + x_{2j} + \cdots + x_{nj}\right) \tag{11}$$

$$STAND_j = \frac{1}{n}\left(\left|x_{1j} - AVERAGE_j\right| + \cdots + \left|x_{nj} - AVERAGE_j\right|\right) \tag{12}$$

$AVERAGE_j = 0$, $x'_{ij} = 0$; $STAND_j = 0$, $x'_{ij} = 0$。

Secondly, the standardized features need to be normalized, as shown in the formula (13):

$$x'_{ij} = \frac{x'_{ij} - \min x}{\max x - x'_{ij}} \tag{13}$$

Where x denotes the value of the original training sample, max (or min) denotes the maximum value for the sample data in the condition of same indicator (or minimum).

3.2 Evaluation Index Experimental Results

This paper uses the following indexes to evaluate experimental results, which are TPR (True Positive Rate), FPR (False Positive Rate), Accuracy, CPU time consumption. They are defined as follows:

TPR = the number of correct results of normal code samples/the actual number of normal code samples,

FPR = the number of malicious code samples which are predicted to be normal code/the actual number of malicious code samples.

3.3 Comparison of Experimental Tesults

Experimental test environment: the platform of Intel Core Duo CPU 2.10GHz and 2.00G RAM's, Matlab v7.11. This paper uses 2000 samples extracted from 10% samples in proportion which contain the 141 attacks recorded test data and additional 14 types of experiments. The experiment designed the AutoEncoder which consists of five layers. The numbers of neurons in the previous four-layer network are 41, 300, 150, 75, respectively. Furthermore, the number of neurons in the last layer is variable, which determine the dimension of data number after dimensionality reduction.

After the pretraining process of the training and testing data, we use AutoEncoder for data dimensionality reduction. Through changing the iterations of the pretraining and fine-tuning, we could get different models, including AutoEncoder + $DBN^{5\text{-}5}$ (pretraining iterations 5 times, fine-tuning5 times); AutoEncoder + $DBN^{10\text{-}10}$ (pre-training iterations 10 times, fine-tuning 10 times); AutoEncoder + $DBN^{10\text{-}5}$ (pre-training iterations 10 times, fine-tuning five times). The detection results of malicious code as shown in Table 1.

Table 1. The Results of the Different Detection Methods

Model	TPR	FPR	Accuracy	CPU time(s)
DBN	95.34%	9.02%	91.4%	1.126

AutoEncoder+DBN$^{5\text{-}5}$	96.79%	15.79%	89.75%	2.625
AutoEncoder+DBN$^{10\text{-}5}$	93.35 %	9.17%	88.95%	1.147
AutoEncoder+DBN$^{10\text{-}10}$	92.20%	1.58%	92.10%	1.243

The experimental results show that with the increase in the number of iterations, in the respect of detection accuracy, the proposed method is superior to the method of single DBN, which was used in the first experiment. Apparently, using AutoEncoder to achieve data dimension reduction is effective, it can improve the detection accuracy, for using AutoEncoder can capture the essential characteristics of date efficiently. Meanwhile, the accuracy of detection (TP) is reduced. Overall, in the respect of prediction accuracy, the mentioned method described in the paper is superior to the single DBN method. It can adapt to the complex environment, achieve effective detection of malicious code, moreover, it consumes less time. Figures 6 and 7 show the error rate in the process of pretraining and fine-tuning. After two iterations, the error rate is maintained at a lower level stably.

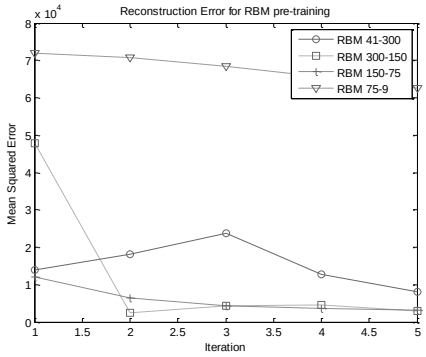

Figure 6. RBM Pretraining Reconstruction Error

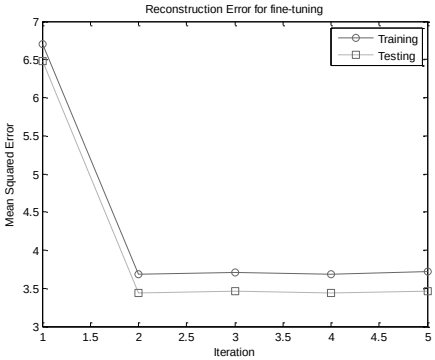

Figure 7. Fine-tuning Reconstruction Error

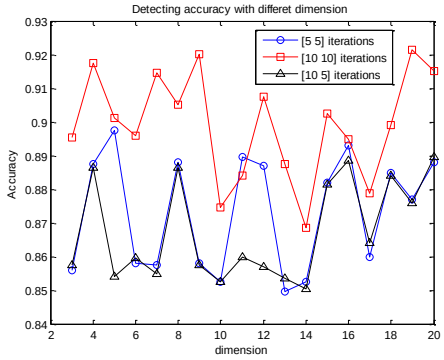

Figure 8. Effect of Dimensions on the Correct Detecting Accuracy

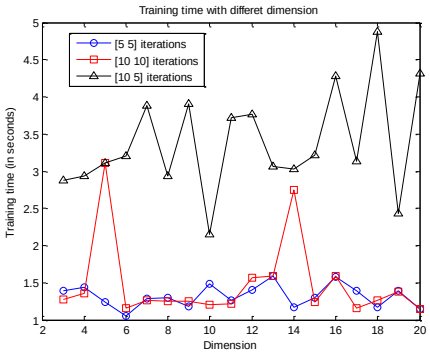

Figure 9. Effect of Dimensions on the Time Consumption

There are many parameters in the AutoEncoder, such as network structure, output dimension of data after dimensionality reduction, the number of iterations for pretraining and fine-tuning, etc. The output dimension of data after dimensionality reduction is one of the major parameters among them. This paper explores the impact of these parameters on these mentioned methods. Figures 8 and 9 respectively show the effect on the detection accuracy and the time consumption of the method. In figure 8, detection accuracy increases with increasing number of iterations. In Figure 9, with the increase of the number of iterations, CPU time consumption varies, but the dimension and training time consumption have no direct correlation, because AutoEncoder can restore data based on less information loss and error.

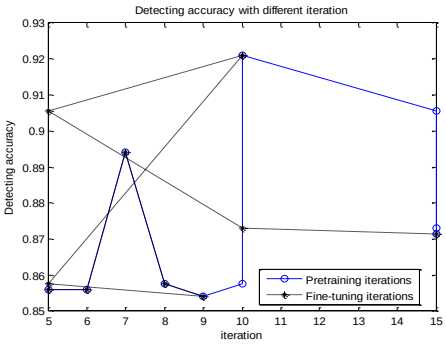

Figure 10. The Relations between the Correct Detecting Accuracy and Iterations

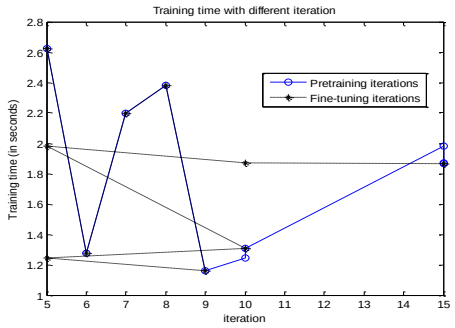

Figure 11. The Relations between the Time Consumption and Iterations

Figure 10 and Figure 11 show the effect on the detection accuracy of the number of iterations and the time consuming. Figure 10 show that when pretraining iterations increased to 10 times, the detection accuracy reached the highest point. Figure 11 shows that when pretraining iterations increased to 10 times, most of the time consumption is maintained at a low level. Fine-tuning process is to adjust the weights using back-propagation, for low-dimensional data, the network is over-learning. The iterations of fine-tuning do not affect two assessed value directly. AutoEncoder reduces the data dimensions and extracts the main features of data through the nonlinear mapping for complex multidimensional data; this makes the effectiveness of the experiment increased when applying DBN to classify. In short, for the detection of malicious code, the hybrid method mentioned in this paper is apparently superior to the single DBN method in the first experiment on the whole.

4. Conclusion

Against the problem of detecting malicious code, we propose a hybrid method of detecting malicious code based on deep learning, which combines the advantages of AutoEncoder and DBN respectively. Firstly, the method used AutoEncoder for data dimensionality reduction to extract the main feature of data. Then the method uses DBN to detect malicious code. Finally, the experiment was verified by KDDCUP'99 dataset. Experimental results show that compared with the detection method using single DBN, the proposed method improves detection accuracy, while reducing the time complexity of the model. However, in practical application, according to actual situation, the method proposed in this paper needs to have further improvements in order to improve its performance.

Acknowledgements

This work was supported in part by The Fundamental Research Funds for the Central Universities (No. 2014MS29).

References

[1] "Symantec Corporation", Symantec Internet security threat report trends for 2010 [EB/OL] (2011-04) [2012-07-01], **(2011)**, http://msisac.cisecurity.org/resources/reports/documents/Symantec, Internet Security Threat report 2010.pdf.
[2] N. Idika and A. P Mathur, "Survey of malware detection technical", Technical Report, Department of Computer Science, Purdue University, **(2007)**.
[3] C. L. Tsai, C. C. Tseng and C. C. Han, Editors, "Intrusive behavior analysis based on honey pot tracking and ant algorithm analysis", 43rd Annual 2009 International Carnahan Conference, **(2009)** October 5-8, Zurich, Switzerland.
[4] W. Wang and I. Murynets, J. Security and Communication Networks, vol. 6, no. 1, **(2013)**.
[5] P. C. Lin, Y. D. Lin, Y. C. Laiand and T. H. Lee, J. Computer Practices, vol. 41, no. 4, **(2008)**.

[6] Y. Sawaya, A. Kubota and Y. Miyake, Editors, "Detection of attackers in services using anomalous host behavior based on traffic flow statistics", IEEE/IPSJ 11th International Symposium, **(2011)** July 18-21, Munich, Bavaria, Germany.
[7] M. Milenkovic, A. Milenkovic and E. Jovanov, J. ACM SIGARCH Computer Architecture News, vol. 33, no. 1, **(2005)**.
[8] M. Christodorescu, S. Jha, S. A. Seshia, D. Song and R. E. Bryant, Editors, Proceedings of the 2005 IEEE Symposium Security and Privacy, **(2005)** May 8-11; Oakland, California.
[9] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns", Wisconsin: University of Wisconsin, **(2006)**.
[10] D. G. Kong, X. B. Tan, H. S. Xi, T. Gong and J. M. Shuai, J. Journal of Software, vol. 22, no. 3, **(2011)**.
[11] A. Shabtai, R. Moskovitch, Y. Elovici and C. Glezer, J. Information Security Technical Report, vol. 14, no. 1, **(2009)**.
[12] Y. X. Ding, X. B. Yuan, D. Zhou, L. Dong and Z. C. An, J. Computers &Security, vol. 30, no. 6, **(2011)**.
[13] R. Wang, D. G. Feng, Y. Yang and P. R. Su, J. Journal of Software, vol. 23, no. 2, **(2012)**.
[14] G. E. Hinton and R. R. Salakhutdinov, J. Science, vol. 313, no. 5786, **(2006)**.
[15] G. E. Hinton, "Distributed representations", Tech. Report, University of Toronto, **(1984)**.
[16] KDDCUP99, Available on, **(2007)**, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Authors

Yuancheng Li, received the Ph.D. degree from University of Science and Technology of China, Hefei, China, in 2003. From 2004 to 2005, he was a postdoctoral research fellow in the Digital Media Lab, Beihang University, Beijing, China. Since 2005, he has been with the North China Electric Power University, where he is a professor and the Dean of the Institute of Smart Grid and Information Security. From 2009 to 2010, he was a postdoctoral research fellow in the Cyber Security Lab, Pennsylvania State University, Pennsylvania, USA. His current research interests include Smart Grid operation and control, information security in Smart Grid.

