

Intrusion Detection Based on Self-adaptive Differential Evolution Extreme Learning Machine with Gaussian Kernel

Junhua Ku^(✉) and Bing Zheng

Department of Information Engineering,
Hainan Institute of Science and Technology, Haikou 571100, China
kujunhua@163.com, 512049181@qq.com

Abstract. In our everyday life, intrusion detection system(IDS) becomes a promising area of research in the domain of security. With the rapid development of network-based services, IDS can detect the intruders who are not authorized to the present computer system, so IDS has emerged as an essential component and an important technique for network security.

In order to conquer the disadvantage of the traditional algorithm for single-hidden layer feedforward neural network (SLFN), an improved algorithm, called extreme learning machine (ELM), is proposed by Huang et al. However, ELM is sensitive to the neuron number in hidden layer and its selection is a difficult-to-solve problem. ELM is an interested area of research for detecting possible intrusions and attacks. In this paper, we propose an improved learning algorithm named self-adaptive differential evolution extreme learning machine with Gaussian Kernel (SaDE-KELM) for classifying and detecting the intrusions. We compare our methods with commonly used ELM, DE-ELM techniques in classifications. Simulation results show that the proposed SaDE-KELM approach achieves higher detection accuracy in classification case.

Keywords: Extreme learning machines · Differential evolution extreme learning machines · Self-adaptive differential evolution extreme learning machines · Intrusion detection · Network security

1 Introduction

Intrusion into computer networks and systems is a major threat in today's network centric world. Few most prevalent intrusion attacks include Denial-of-Service (DoS) attacks, Distributed-Denial-of-Service(DDoS) attacks, probing based attacks and account takeover attacks. Intrusion detection identifies computer attacks by observing various records processed on the network. Intrusion detection models are classified into two variants, misuse detection and anomaly detection systems. Misuse detection can discover intrusions based on a known pattern also known as signatures [1]. Anomaly detection can identify the malicious activities by observing the deviation from normal network traffic pattern [2]. Hence anomaly detection can identify new anomalies. The difficulty with the current developmental techniques is not only the high false positive rate, but also the low false negative rate.

Recently, extreme learning machine (ELM) [3, 4], was developed to improve the efficiency of SLFNs. ELM is different from the conventional learning algorithms for neural networks (such as BP algorithms [5]), which can easily deal with difficulties in manually tuning control parameters (learning rate, learning epochs, etc.) and/or local minima. ELM is fully automatically implemented without iterative tuning, and tends to provide better generalization performance at extremely fast learning speed. Furthermore, It was popular for its fast training speed by means of utilizing random hidden node parameters and calculating the output weights with least square algorithm [6–10]. However, in ELM, the number of hidden nodes is assigned in advance, the hidden node parameters are randomly chosen and they stay the same during the training phase. Many non-optimal nodes may exist and contribute less in minimizing the cost function. Moreover, in [11] Huang et al. indicated that ELM was inclined to require more hidden nodes than conventional tuning-based algorithms [12, 13] in many cases.

Differential evolution (DE) [14] which is a simple but powerful population-based stochastic direct searching technique is a frequently used method for selecting the network parameters [15–17]. In [15], DE is directly adopted as a training algorithm for feed forward networks where all the network parameters are encoded into one population vector and the error function between the network approximate output and the expected output is used as the fitness function to evaluate all the populations. However, Subudhi and Jena [16] have indicated that using the DE approach alone for the network training may lead to a decrease in convergence. Therefore, in [17], DE-ELM based on DE and ELM has been developed for SLFNs. Using the DE method to optimize the network input parameters and the ELM algorithm to calculate the network output weights, DE-ELM has shown several promising features. It not only ensures a more compact network size than ELM, but also has better generalization performance.

However, in the above DE based neural network training algorithms, the trial vector generation strategies and the control parameters in DE have to be manually chosen. Therefore, we propose the SaDE-KELM for SLFNs. In SaDE-KELM, the hidden node learning parameters are optimized by the self-adaptive differential evolution algorithm. We verify our SaDE-KELM using the data originated from the 1998 DARPA Intrusion Detection Evaluation Program 1999 [18], and deliberate to regard as a common benchmark for evaluating intrusion detection techniques [19–21].

The rest of the paper is organized as follows. In Sect. 2, a brief introduction to ELM and SaDE are given. In Sect. 3, we introduce model of proposed SaDE-ELM algorithm in detail. In Sect. 4, we present the dataset we use in our SaDE-KELM-based intrusion detection technique. In Sect. 5, we ran experiments for detecting intrusion in network traffic data and obtained the performance comparisons among ELM-based techniques, DE-ELM-based techniques and SaDE-KELM techniques. In Sect. 6, we conclude and summarize our results.

2 Background

As we know that ELM is very efficient and effective for SLFNs' training algorithms. In this section, we gave a review of ELM in brief. At the same time we briefly reviewed SaDE approach and proposed our SaDE-ELM techniques in the end.

2.1 Extreme Learning Machine (ELM)

For N arbitrary distinct samples (x_j, t_j) , where $x_j = [x_{j1}, x_{j2}, \dots, x_{jm}]^T \in \mathbb{R}^n$, $t_j = [t_{j1}, t_{j2}, \dots, t_{jm}]^T \in \mathbb{R}^m$, SLFNs with L hidden nodes and activation function $g(x)$ are

$$\sum_{i=1}^L \beta_i g_i(x_j) = \sum_{i=1}^L \beta_i g_i(w_i \cdot x_j + b_j) = o_j \quad (j = 1, 2, \dots, N) \quad (1)$$

where $w_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T$ is the weight vector connecting the i th hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ is the weight vector connecting the i th hidden node and the output nodes, b_i is the threshold of the i th hidden node, $w_i \cdot x_j$ denotes the inner product of w_i and x_j , $g(x)$ is activation function and Sigmoid, Sine, Hardlim and other functions are commonly used. The output nodes are chosen linear in this paper, and $o_j = [o_{j1}, o_{j2}, \dots, o_{jm}]^T$ is the j th output vector of the SLFNs [22].

The SLFNs with L hidden nodes and activation function $g(x)$ can approximate these N samples with zero error. It means $\sum_{j=1}^L \|o_j - t_j\| = 0$ and there exist β_i , w_i and b_i such that

$$\sum_{i=1}^L \beta_i g_i(x_j) = \sum_{i=1}^L \beta_i g_i(w_i \cdot x_j + b_j) = t_j \quad (j = 1, 2, \dots, N) \quad (2)$$

The equation above can be expressed compactly as follows:

$$\mathbf{H}\beta = \mathbf{T} \quad (3)$$

where $\mathbf{H}(w_1, w_2, \dots, w_L, b_1, b_2, \dots, b_L, x_1, x_2, \dots, x_N)$

$$= [h_{ij}] \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & g(w_1 \cdot x_1 + b_2) & \cdots & g(w_1 \cdot x_1 + b_L) \\ g(w_1 \cdot x_2 + b_1) & g(w_2 \cdot x_2 + b_2) & \cdots & g(w_L \cdot x_2 + b_L) \\ \vdots & \vdots & & \vdots \\ g(w_1 \cdot x_N + b_1) & g(w_2 \cdot x_N + b_2) & \cdots & g(w_L \cdot x_N + b_L) \end{bmatrix}_{N \times L}$$

$$\beta = \begin{pmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1m} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{L1} & \beta_{L2} & \cdots & \beta_{Lm} \end{pmatrix} \text{ and } \mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ t_{N1} & t_{N2} & \cdots & t_{Nm} \end{pmatrix}$$

We call the matrix \mathbf{H} the hidden layer output matrix of the neural network. Meanwhile, with respect to inputs x_1, x_2, \dots, x_N , it is easy to see that the i th hidden node output is the i th column of \mathbf{H} .

Because L is far smaller than N in actual problem, here in particular it is rare condition that L equals to N [23]. That is to say, in ELM algorithm, the important thing is to find a least-square solution of this linear system as follow

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T} \quad (4)$$

where \mathbf{H}^\dagger is the Moore-Penrose Generalized inverse of matrix \mathbf{H} , depending on the singularity of $\mathbf{H}^T \mathbf{H}$ or $\mathbf{H} \mathbf{H}^T$, we can obtain $\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T$ or $\mathbf{H}^T (\mathbf{H} \mathbf{H}^T)^{-1}$. Then the output function of ELM can be modeled as follows.

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta = \mathbf{h}(\mathbf{x})\mathbf{H}^\dagger \mathbf{T} \quad (5)$$

It's getting more stable to introduce a positive coefficient into ELM. For example, $\mathbf{H}^T \mathbf{H}$ is nonsingular, the coefficient $1/C$ is added to the diagonal of $\mathbf{H}^T \mathbf{H}$ in the calculation of the output weights β . So, the corresponding function of the regularized ELM is:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta = \mathbf{h}(\mathbf{x})\mathbf{H}^T \left(\frac{E}{C} + \mathbf{H} \mathbf{H}^T \right)^{-1} \mathbf{T} \quad (6)$$

Moreover, we know that many nonlinear activation and kernel functions can be used in ELM. Let $\Omega_{\text{ELM}} = \mathbf{H} \mathbf{H}^T : \Omega_{\text{ELM}i,j} = \mathbf{h}(\mathbf{x}_i)\mathbf{h}(\mathbf{x}_j) = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$, the output function can be written as:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\mathbf{H}^T \left(\frac{E}{C} + \mathbf{H} \mathbf{H}^T \right)^{-1} \mathbf{T} = \begin{bmatrix} \mathbf{K}(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ \mathbf{K}(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}^T \left(\frac{E}{C} + \Omega_{\text{ELM}} \right)^{-1} \mathbf{T} \quad (7)$$

The hidden layer feature mapping $\mathbf{h}(\mathbf{x})$ need not to be known, and instead its corresponding kernel $\mathbf{K}(\mathbf{u}, \mathbf{v})$ can be computed. In this way, the Gaussian kernel is used, $\mathbf{K}(\mathbf{u}, \mathbf{v}) = \exp(-\gamma \|\mathbf{u} - \mathbf{v}\|^2)$ [14].

2.2 Self-adaptive Differential Evolution

Differential evolution (DE), proposed by Storn and Price in 1995, is a powerful evolutionary algorithm [24]. There are three parameters in DE, which are the population size NP , mutation scaling factor F and crossover rate CR . To overcome the limitations of choosing the parameters in DE, Brest et al. [25] proposed a parameter adaptation technique to choose the mutation scaling factor F and crossover rate CR namely SaDE which performs better than the basic DE. In general, SaDE is composed of three main steps: mutation, crossover, and selection [26].

We consider the following optimization problem:

$$\text{Minimize } f(\mathbf{x}), \mathbf{x}_i \in R_D$$

where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iD}]^T, i = 1, 2, \dots, NP$ is a target vector of D decision variables. During the mutation operation, mutant vector \mathbf{v}_i is generated by mutation strategy in the current population:

$$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3}) \quad (8)$$

where r_1, r_2, r_3 are randomly mutually exclusive integers in the range $[1, NP]$, and $r_1 \neq r_2 \neq r_3 \neq i$.

Following mutation, trial vector u_i is generated between x_i and v_i during crossover operation where the most widely used operator is the binomial crossover performed as follows:

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } (\text{rndreal}(0, 1) < CR \text{ or } j = j_{\text{rand}}), \\ x_{ij}, & \text{otherwise} \end{cases} \quad (9)$$

where j_{rand} is a integer randomly chosen in the range $[1, D]$, and $\text{rndreal}(0, 1)$ is a real number randomly generated in $(0, 1)$. Finally, to keep the population size constant during the evolution, the selection operation is used to determine whether the trial or the target vector survives to the next generation according to one-to-one selection:

$$x_i = \begin{cases} u_i, & \text{if } (f(u_i) < f(x_i)) \\ x_i, & \text{otherwise} \end{cases} \quad (10)$$

where $f(x)$ is the optimized objective function. During the evolution, F and CR are adaptively tuned to improve the performance of DE for each individual

$$F_{i,G+1} = \begin{cases} F_l + \text{rand}_1 \cdot F_u & \text{if } (\text{rand}_2 < \tau_1) \\ F_{i,G} & \text{otherwise} \end{cases} \quad (11)$$

$$CR_{i,G+1} = \begin{cases} \text{rand}_3 & \text{if } (\text{rand}_4 < \tau_2) \\ CR_{i,G} & \text{otherwise} \end{cases} \quad (12)$$

where $F_{i,G+1}$ and $CR_{i,G+1}$ are the mutation scaling factor and crossover rate for i individual in G generation respectively, $\text{rand}_j = 1;2;3;4$ are randomly chosen from $(0, 1)$, τ_1 and τ_2 both valued 0.1 which is used to control the generation of F and CR , F_l valued 0.1 and F_u is valued 0.9. In the first generation, F and CR are initialized to 0.5.

3 Model of Proposed SADE-KELM Algorithm

Since the ELM may not reach the optimal result in classification or regression, a hybrid approach integrated SaDE and ELM with gaussian kernel namely SADE-KELM algorithm to optimize the input weights and hidden biases is able to obtain better generalization performance than ELM. In SaDE-KELM, we proposed SaDE-KELM for SLFNs by incorporating the SaDE to optimize the network input weights and hidden node biases and ELM with gaussian kernel to derive the network output weights.

Given a set of training data and L hidden nodes with an activation function $g(\cdot)$, we summarize the SADE-KELM algorithm in the following steps.

Step 1. Initialization

A set of NP vectors where each one includes all the network hidden node parameters are initialized as the populations of the first generation

$$\theta_{k,G} = [w_{1,k,G}^T, \dots, w_{L,k,G}^T, b_{1,k,G}^T, \dots, b_{L,k,G}^T] \quad (13)$$

where w_j and b_j ($j = 1, \dots, L$) are randomly generated, G represents the generation and $k = 1, 2, \dots, NP$.

Step 2. Calculations of output weights and RMSE

Calculate the network output weight matrix and root mean square error (RMSE) with respect to each population vector with the following equations, respectively.

$$\beta_{k,G} = H_{k,G}^\dagger T \quad (14)$$

$$RMSE_{k,G} = \sqrt{\frac{\sum_{i=1}^N \left\| \sum_{j=1}^L \beta_j g(w_{j,k,G}, b_{j,k,G}, x_i) - t_i \right\|^2}{m \times N}} \quad (15)$$

Then use the value of RMSE to calculate the new best population vector $\theta_{k,G+1}$ with the following equation.

$$\theta_{k,G+1} = \begin{cases} u_{k,G+1} & \text{if } (RMSE_{\theta_{k,G}} - RMSE_{u_{k,G+1}}) > \varepsilon \cdot RMSE_{\theta_{k,G}} \\ u_{k,G+1} & \text{if } |RMSE_{\theta_{k,G}} - RMSE_{u_{k,G+1}}| < \varepsilon \cdot RMSE_{\theta_{k,G}} \\ & \text{and } \|\beta_{u_{k,G+1}}\| < \|\beta_{\theta_{k,G}}\| \\ \theta_{i,G} & \text{otherwise} \end{cases} \quad (16)$$

where ε is the preset small positive tolerance rate. In the first generation, the population vector with the best RMSE is stored as $\theta_{\text{best},1}$ and $RMSE_{\theta_{\text{best},1}}$.

All the trial vectors $u_{k,G+1}$ generated at the $(G + 1)$ th generation are evaluated using Eq. (11). The norm of the output weight $\|\beta\|$ is added as one more criteria for the trial vector selection as pointed out in [23] that the neural networks tend to have better generalization performance with smaller weights.

The three operations mutation, crossover and selection are repeated until the preset goal is met or the maximum learning iterations are completed. At last we calculate the output weigh $\beta = [\beta_{i1} \beta_{i2} \dots \beta_{iL}]^T$ with equation $\beta = H^\dagger T$.

4 Intrusion Detection Using SADE-KELM

In this section, we describe the dataset that we use for our numerical studies, and our SaDE-KELM approach to classification of intrusions in the data.

4.1 Dataset Description

The dataset we use is from the 1998 DAPRA intrusion detection program [27]. Four main categories of attacks were simulated:

- (1) DoS: denial of service attack
- (2) R2L: unauthorized access from a remote machine
- (3) U2R: unauthorized access to local root privileges
- (4) Probing: surveillance and other probing

In the intrusion detection simulation, the dataset was labeled with 22 attack types falling into the four categories shown in Table 1. The feature list and its descriptions are in Tables 2, 3 and 4.

Table 1. Attack type

Denial of Service	User to Root	Remote to User	Probing
Back	Perl	FTP Write	IP Sweep
Neptune	Buffer Overflow	Guess Password	Nmap
Land	Load Module	Imap	Port Sweep
Teardrop	Rootkit	Multihop	Satan
Ping of Death		Phf	
Smurf		Spy	
		Warezcclient	
		Warezmater	

Table 2. Basic features of individual tcp connections

Feature name	Description	Type
Duration	Length (number of seconds) of the connection	Continuous
protocol_type	Type of the protocol	Discrete
service	Network service on the destination	Discrete
src_bytes	Number of data bytes from source to destination	Continuous
dst_bytes	Number of data bytes from destination to source	Continuous
flag	Normal or error status of the connection	Discrete
land	1 if connection is from/to the same host/port, 0 otherwise	Discrete
wrong_fragment		Continuous
urgent	Number of “wrong” fragments	Continuous
	Number of urgent packets	

4.2 Intrusion Detection System Using SaDE-ELM

We use a ELM method to classify the data to provide a comparison benchmark, our SaDE-KELM IDS has the following steps.

Table 3. Content features within a connection suggested by domain knowledge

Feature name	Description	Type
Hot	Number of “hot” indicators	Continuous
Num failed logins	Number of failed login attempts	Continuous
Logged in	1 if successfully logged in, 0 otherwise	Discrete
Num compromised	Number of “compromised” conditions	Continuous
Root shell	1 if root shell is obtained, 0 otherwise	Discrete
Su attempted	1 if “su root” command attempted, 0 otherwise	Discrete
Num root	Number of “root” accesses	Continuous
Num file creations	Number of file creation operations	Continuous
Num shells	Number of shell prompts	Continuous
Num access files	Number of operations on access control files	Continuous
Num outbound cmds	Number of outbound commands in an ftp session	Continuous
Is hot login	1 if the login belongs to the “hot” list, 0 otherwise	Discrete
Is guest login	1 if the login is a “guest”login, 0 otherwise	Discrete

Table 4. Traffic features computed using a two-second time window

Feature name	Description	Type
Count	Number of connections to the same host as the current connection in the past two seconds	Continuous
Error rate	% of connections that have “SYN” errors	Continuous
Error rate	% of connections that have “REJ” errors	Continuous
Same srv rate	% of connections to the same service	Continuous
Diff srv rate	% of connections to different services	Continuous
Srv count	number of connections to the same service as the current connection in the past two seconds	Continuous
Srv error rate	% of connections that have “SYN” errors	Continuous
Srv error rate	% of connections that have “REJ” errors	Continuous
Srv diff host rate	% of connections to different hosts	Continuous

Step 1. Converting the raw TCP/IP dump data into machine readable form.

Step 2. SaDE-KELM and ELM are trained on normal data and different types of attacks. For the binary classification case, the data has 41 features and falls into 2 classes: normal and attack; for the multi-class classification case, the data has 41 features and falls into 23 classes: normal and 22 types of attack. The model is trained in a large program which can test immediately after the training completed. According to SaDE-KELM theory that has been introduced above, we can summarize the following steps.

For N arbitrary distinct samples (x_i, t_i) , $i = 1, \dots, N$, and hidden nodes and activation function $g(x)$:

- (2.1) A set of NP individual parameter vectors $\theta_{k,G}$ ($k = 1, 2 \dots NP$), where each one includes all the network hidden node parameters are initialized as the populations of the first generation;
- (2.2) In the case of $g(x)$ and L are invariable run the three operations including mutation, crossover and selection to produce the new population, and the process is repeated until the stop condition is completed.
- (2.3) Changing the type of $g(x)$ and increase the number of hidden nodes L gradually from one to find the most suitable $g(x)$ and L to construct an optimal forecasting model with the best testing accuracy;
- (2.4) Calculating the output matrix H ;
- (2.5) Calculating the output weights $\beta = H^\dagger H$.

Step 3. Testing phase: ELM, DE-ELM and SaDE-KELM are used to predict the type of each data point in the testing dataset, and their performances are compared.

Experiments have shown that if the number of values in an attribute is not too large, this coding is more stable than using a single number. The simulation of the three algorithms on all datasets are carried out using MATLAB 2013a on a machine with an Intel Core 2 Duo, 2.26 GHz CPU and 4 GB RAM.

5 Simulation Results

The datasets being tested are 2000, 4000, 8000 connection data chosen randomly from the dataset downloaded from the website [18]. We split them equally into training data and testing data. Simulation results including average testing accuracy and corresponding 95% confidence interval are given in Table 4.

In order to test the relationship between SaDE-KELM and the number of hidden layer, according to the different number of hidden layer nodes, we made classification tests using ELM, DE-ELM and SaDE-KELM respectively. Simulation results are given in Table 5.

Table 5. Performance comparison results

Dataset Size	ELM		DE-ELM		SaDE-KELM	
Training/Testing	Accuracy (%)	95% Confidence Interval (%)	Accuracy (%)	95% Confidence Interval (%)	Accuracy (%)	95% Confidence Interval
1000/1000	99.32	99.08–99.47	99.33	99.15–99.51	99.55	99.05–99.65
2000/2000	99.10	98.82–99.23	99.24	98.90–99.44	99.47	99.25–98.58
4000/4000	99.07	98.79–9.28	99.18	99.01–99.26	99.35	99.11–99.65

Figure 2 show the time spent by ELM, DE-ELM and SaDE-KELM when testing the same size of dataset. It can be seen that the training time and testing time spent by SaDE-KELM increase sharply when the size of data increases. In comparison, ELM

and DE-ELM increase slowly when the number of data increases. Eventually, DE-ELM starts consuming more time for both training and testing than ELM.

Prediction of test samples (SaELM) with $N_{min} = 5$, $N_{max} = 120$ and $N_{Interval} = 5$ for the classification problem can be seen from Fig. 1. A clear time consumption comparison can be seen from Fig. 2. From the results, we can conclude that ELM performs better than DE-ELM and SaDE-KELM in terms of speed. To increase accuracy, we can implement SaDE-KELM. This shows that our proposed SaDE-ELM methods have better scalability than ELM and DE-ELM when classifying network traffic for intrusion detection.

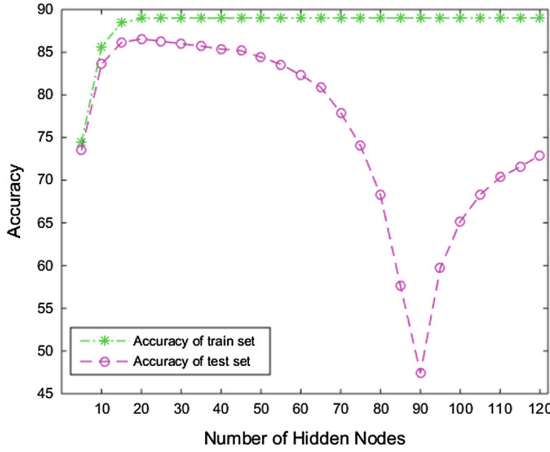


Fig. 1. Prediction of test samples (SaDE-KELM) with $N_{min} = 5$, $N_{max} = 120$ and $N_{Interval} = 5$ for the classification problem.

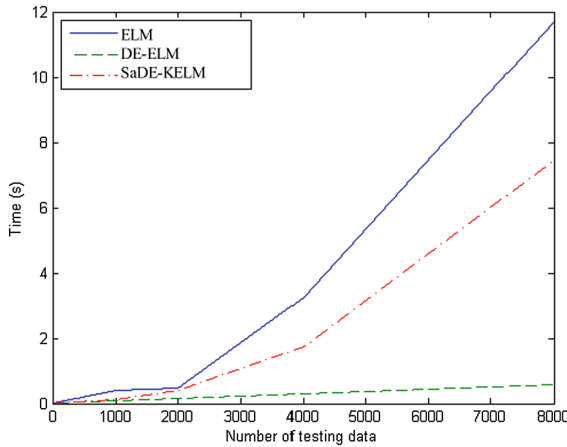


Fig. 2. Testing time comparison

6 Conclusion

ELM is sensitive to the neuron number in hidden layer and its selection is a difficult-to-solve problem. Recently, Extreme Learning Machine has been widely applied in IDS. It's an important and efficient way to improve the performance of IDS. However, the conventional feedback relevance schemes could not give considerations to the both accuracy and speed. To combine KELM(Extreme Learning Machine with Gaussian Kernel) with the SaDE method, we can overcome the limitation of the conventional problems. In this paper, we have made a comparison by the use of ELM, DE-ELM and SaDE-KELM for intrusion detection. For the SaDE-KELM, By incorporating the SaDE to optimize the network hidden node parameters and employing the KELM to derived the network output weights. Obviously, the proposed SaDE-KELM can obtain higher accuracy. Simulation results show that the proposed SaDE-KELM approach achieves higher detection accuracy in classification case.

Acknowledgement. This research was supported by key science research project of Education Department of Hainan province (Hnky2017ZD-20).

References

1. Ilgun, K., Kemmerer, R.A., Porras, P.A.: State transition analysis: a rule-based intrusion detection approach. *IEEE Trans. Softw. Eng.* **21**(3), 181–199 (1995)
2. Ikram, S.T., Cherukuri, A.K.: Improving accuracy of intrusion detection model using PCA and optimized SVM[J]. *CIT. J. Comput. Inf. Technol.* **24**(2), 133–148 (2016)
3. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: *Proceedings of International Joint Conference on Neural Networks (IJCNN2004)*, vol. 2, no. 25–29, pp. 985–990
4. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. *Neurocomputing* **70**(1–3), 489–501
5. Espana-Boquera, S., Zamora-Martínez, F., Castro-Bleda, M.J., et al.: Efficient BP algorithms for general feedforward neural networks. In: *International Work-Conference on the Interplay Between Natural and Artificial Computation* (pp. 327–336). Springer, Heidelberg (2007)
6. Thatte, G., Mitra, U., Heidemann, J.: Parametric methods for anomaly detection in aggregate traffic. *IEEE/ACM Trans. Netw.* **19**(2), 512–525 (2011)
7. Qin, M., Hwang, K.: Frequent episode rules for internet anomaly detection. In: *Proceedings of the Network Computing and Applications, Third IEEE International Symposium*. Washington, DC, USA: IEEE Computer Society, pp. 161–168 (2004)
8. He, X., Papadopoulos, C., Heidemann, J., Mitra, U., Riaz, U.: Remote detection of bottleneck links using spectral and statistical methods. *Comput. Netw.* **53**, 279–298 (2009)
9. Streilein, W.W., Cunningham, R.K., Webster, S.E.: Improved detection of low-profile probe and denial-of-service attacks. In: *Proceedings of the 2001 Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection* (2001)
10. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**, 273–297 (1995)
11. Huang, G.-B., Wang, D.H., Lan, Y.: Extreme learning machines: a survey. *Int. J. Mach. Learn. Cybern.* **2**(2), 107–122 (2011)
12. Tandon, G.: Weighting versus pruning in rule validation for detecting network and host anomalies. In: *Proceedings of the 13th ACM SIGKDD international*. ACM Press (2007)

13. Liao, Y., Vemuri, V.R.: Use of k-nearest neighbor classifier for intrusion detection. *Comput. Secur.* **25**, 439–448 (2002)
14. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **11**(4), 341–359 (2004)
15. Ilonen, J., Kamarainen, J.I., Lampinen, J.: Differential evolution training algorithm for feedforward neural networks. *Neural Process. Lett.* **17**, 93–105 (2003)
16. Subudhi, B., Jena, D.: Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification. *Neural Process. Lett.* **27**, 285–296 (2008)
17. Zhu, Q.-Y., Qin, A.-K., Suganthan, P.-N., Huang, G.-B.: Evolutionary extreme learning machine. *Pattern Recog.* **38**(10), 1759–1763 (2005)
18. KDDCUPdataset: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (1999)
19. Mukkamala, S., Sung, A.: Detecting denial of service attacks using support vector machines. In: *Proceedings of the 12th IEEE International Conference on Fuzzy Systems* (2003)
20. Luo, M., Wang, L., Zhang, H., Chen, J.: A research on intrusion detection based on unsupervised clustering and support vector machine. In: Qing, S., Gollmann, D., Zhou, J. (eds.) *ICICS 2003*. LNCS, vol. 2836, pp. 325–336. Springer, Heidelberg (2003). doi:[10.1007/978-3-540-39927-8_30](https://doi.org/10.1007/978-3-540-39927-8_30)
21. Kim, D.S., Park, J.S.: Network-Based Intrusion Detection with Support Vector Machines. In: Kahng, H.-K. (ed.) *ICOIN 2003*. LNCS, vol. 2662, pp. 747–756. Springer, Heidelberg (2003). doi:[10.1007/978-3-540-45235-5_73](https://doi.org/10.1007/978-3-540-45235-5_73)
22. Lin, Y., Lv, F., Zhu S., Yang, M., Cour, T., Yu, K., Cao, L., Huang, T.S.: Large-scale image classification: fast feature extraction and SVM training. In: *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1689–1696 (2011)
23. Huang, G.-B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Netw.* **17**(4), 879–892 (2006)
24. Storn, R., Price, K.: Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.* **11**, 341–359 (1997)
25. Brest, J., Greiner, S., Bösković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: A comprehensive study on numerical benchmark problems. *IEEE Trans. Evol. Comput.* **10**, 646–657 (2006)
26. Wu, J., Cai, Z.H.: Attribute weighting via differential evolution algorithm for attribute weighted naive bayes (WNB). *J. Comput. Inf. Syst.* **7**, 1672–1679 (2011)
27. Stolfo, S., Fan, W., Lee, W., Prodromidis, A., Chan, P.K.: Costbased modeling for fraud and intrusion detection: results from the JAM project. In: *Proceedings of DARPA Information Survivability Conference and Exposition*, vol. 2, pp. 130–144 (2002)

Parallel Architecture, Algorithm and Programming
8th International Symposium, PAAP 2017, Haikou,
China, June 17-18, 2017, Proceedings
Chen, G.; Shen, H.; Chen, M. (Eds.)
2017, XV, 629 p. 299 illus., Softcover
ISBN: 978-981-10-6441-8