Neural Network Based Intrusion Detection Systems with Different Training Functions

Gozde Karatas
Istanbul Kultur University
Mathematics and Computer Sciences Department,
34158, Istanbul, Turkey
g.karatas@iku.edu.tr

Ozgur Koray Sahingoz
Istanbul Kultur University
Computer Engineering Department,
34158, Istanbul, Turkey
o.sahingoz@iku.edu.tr

***Abstract*—** In the last decades, due to the improvements in networking techniques and the increased use of the Internet, the digital communications entered all of the activities in the global marketplace. Parallel to these enhancements the attempts of hackers for intruding the networks are also increased. They tried to make unauthorized access to the networks for making some modifications in their data or to increase the network traffic for making a denial of service attack. Although a firewall seems as a good tool for preventing this type of attacks, intrusion detection systems (IDSs) are also preferred especially for detecting the attack within the network system. In the last few years, the performance of the IDS is increased with the help of machine learning algorithms whose effects depend on the used training/learning algorithm. Mainly it is really hard to know which learning algorithm can be the fastest one according to the problem type. The algorithm selection depends on lots of factors such as the size of data sets, number of nodes network design, the targeted error rate, the complexity of the problem, etc. In this paper, it is aimed to compare different network training function in a multi-layered artificial neural network which is designed for constructing an effective intrusion detection system. The experimental results are depicted in the paper by explaining the efficiency of the algorithms according to their true-positive detection rates and speed of the execution.

Keywords—network security; intrusion detection system; neural networks; training functions

I. INTRODUCTION

In the last few years, Internet technology has extended its application area in many different domains in our life such as banking operations, online auctions, electronic commerce applications, social networking, and online application/ registration, etc. However, due to the weakness of computer systems' security, lots of computer network has often been intruded by the hackers especially with denial-of-service or distributed denial-of-service attacks. Use of firewalls and some access control mechanism are one of the most preferred intrusion detection/prevention techniques. However, they are also failed if the intruder is within the network, or he is sophisticated and uses previously unknown attack types. Therefore, implementation of an intrusion detection system (IDS) which can adapt itself to the new types of attacks is a very trivial task. IDS can be located in the network as depicted in Figure 1 to control either intranet communication or internet communication by coordinating with firewall and network security admin.

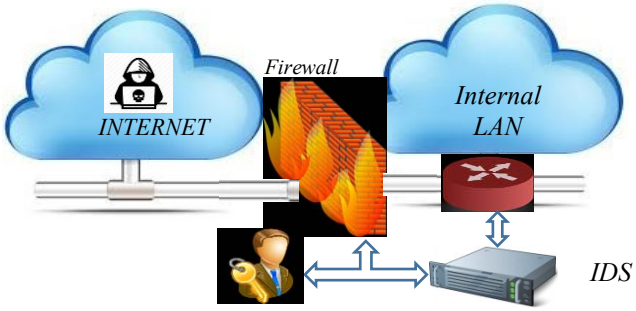

Fig. 1. Intrusion Detection System in a Network

One of the biggest and most challenging issues in network security is distributed denial-of-service (DDoS) attacks [1, 2]. There are lots of request to the network server and determining whether or not it is from a human. Because, bots that attack to the servers generally don't behave like humans. To detect these type of attacks, use of IDS is mostly preferred. IDS is not a problem for standard computer networks but also it is accepted as a great problem for different networking domains which are arised according to technological developments. Cloud computing is a popular paradigm and in these platforms attacks results very important processing power loss [3]. Additionally, in a wireless sensor network concept, the conservation of energy is very important. To increase the lifetime of the system, detecting the intrusions and also prevention of them are very important [4].

According to the data analysis concept an IDS system can be divided into two classes: ***Signature-based IDS*** maintains a database of previously known attacks by storing their signatures. When an attack occurs, then the IDS compares the sniffed data with the data stored in signature database. If the signatures are matched, then the system produces an attack signals to the network security admins. This type of IDS is very efficient and fast if the attack type is previously known, However, it is a trivial task to maintain up-to-dateness of the signature database. Therefore, signature based IDSs are inefficient to detect zero day attacks. ***Anomaly-based IDS*** sniffs the network and if they

978-1-5386-3449-3/18/$31.00 ©2018 IEEE

encounter with an anomalous behavior, which is defined according to the previous activities in the network system, they produce an alarm signal. The main advantage of the system is about the ability to detect the zero day attacks.

Anomaly detection mainly depends on the representation of the normal behavior of the users, hosts and network connections. It is really hard to detect abnormal requests in the network. Therefore, machine learning algorithms are used as a flexible and powerful technique. There are many works in the literature, which covers the advantages of the use of neural networks [5-8]. They have a different type of implementation in a different programming environment and different network design. Due to these differences, they have different performances. Although, most of the researchers preferred the use of *single hidden layer* in their neural network design, due to the increase in the speed of computers, and especially emerging technologies in the parallel computation, multi-layer perceptron, which have the main disadvantage on their slow training time, and requirement of lots of data, are also started for efficiently usage.

One of the important parameters in the development of ANN-based IDS systems is the selection of the training functions. There are a number of training functions in the literature. In addition, a researcher can implement his own training function on his own. In this paper, it is aimed to compare most preferred neural network training functions (as *trainbr, trainc, traincgp, trainlm, trainoss, trainr and trainscg)* in the literature and giving comparative results between them in the application environment of intrusion detection systems which is implemented with a multi-layered neural network approach. This comparison implemented in two different dimensions. First important one is the accuracy rate, which shows the efficiency and the effectiveness of the training function. Second, one is about the execution time of the function, which is important for calculating the training time of the system.

The rest of the paper is organized as follows. In Section II, the necessary information about training functions is given. After that, the details of the proposed system are explained in Section III. An experimental is framework is conducted, and results are drawn in Section IV. Finally, in Section V conclusion and future works are depicted.

II. Training Functions of ANN

The working principle of an artificial neural network (ANN) is based on a supervised learning approach from previous experiences of the system by obtained a labeled data as a training set. After this training process, the success of the system depends on the performance of the test data. And this performance depends on different network design criterion. One of the most important factor to effect the performance of the system is the selection of the Training Function. In the Literature, there are a number of training functions. The effect of the selected function mainly depends on the size of the data, the capability of the computer (especially according to memory and processor consideration), application environment, etc.

Some of the mostly preferred training functions are shortly named as *trainc, trainlm, trainbfg, trainscg, traincgp, trainoss, trainbr trainr*. These functions are detailed in the following part.

Trainc is a training function that updates weight and bias values according to learning rules with incremental updates after each presentation of an input. Inputs are presented in periodical order. For each epoch, each vector (or sequence) is submitted in order to the network, with the weight and bias values updated accordingly after each individual submission [19].

Trainlm is a training function that updates weight and bias values by using Levenberg-Marquardt Optimization. Levenberg-Marquardt is a method that significantly improves learning speed by using the steepest descent landing method and the Newton Algorithm, which requires a second derivative [10]. Each variable is calculated as in (1).

$$jj = J_X * J_X$$
$$J_E = J_X * E \quad (1)$$
$$d_X = -(jj + I * m_u)/J_E$$

where E is all errors, I is the identity matrix, J_X is Jacobian, m_u is adaptive value and X is weight and bias variables[11, 12]. Trainlm is one of the fastest backpropagation algorithm in the system, so it is often preferred in supervised learning. Disadvantage over the other training functions is that it requires too much memory.

Trainbfg is a training function that updates weight and bias values using BFGS quasi-Newton method. BFGS performs a series of operations to reduce the Hessian matrix in the quasi-Newton method [10]. Each variable is calculated as in (2).

$$X = X + m * d_X \quad (2)$$

Where X weight and bias variables, d_x is the search direction and m is selected to minimize the performance. In order for the iterations to be successful, the search direction is computed as in (3).

$$d_X = -H/g_X \quad (3)$$

where g_x is the gradient and H is a approximate Hessian matrix [13]. Trainbfg can train on networks that have values with derivative functions. These values are its weight, net input, and transfer functions.

Trainscg is a training function that updates weight and bias values using the scaled conjugate gradient method. The Scaled Conjugate Gradient Method is a supervised learning algorithm for neural networks, and is a member of Conjugate Gradient Methods. Conjugate Gradient Method iteratively try to reach the minimum value, like backpropagation. But unlike backpropagation, Conjugate gradient method will proceed in a direction which is conjugate to the previous steps[14].

Traincgp is a training function that updates weight and bias values using conjugate gradient backpropagation with Polak-Ribiére updates. Another method of the Conjugate Gradient Algorithm was proposed by Polak and Ribiére. The search direction at each iteration is determined by the formula (4).

978-1-5386-3449-3/18/$31.00 ©2018 IEEE

$$p_k=-g_k+\beta_k+p_{k-1}$$

$$\beta_k=\Delta g_{k-1}^T g_k / g_{k-1}^T g_{k-1} \quad (4)$$

where p_k is new search direction and g_k is gradient. Each variable is calculated as in (5).

$$X=X+m*d_X \quad (5)$$

where d_X is the search direction, m is selected to minimize the performance and X weight and bias variables. The first search direction is the negative of the gradient of performance. In order for the iterations to be successful, the search direction is computed from the new gradient and the previous search direction as depicted in (6).

$$d_X=-g_X+d_{X\_old}*Z \quad (6)$$

where g_X is the gradient and Z is parameter. For the Polak-Ribiére variation Z is computed as in (7).

$$Z=\left(\left(g_X-g_{X_{old}}\right)' * g_X\right)/NS \quad (7)$$

where NS is the norm square of the previous gradient, and $g_{X_{old}}$ is the gradient on the previous iteration[15, 16].

Trainoss is a training function that updates weight and bias values using the one-step secant method. The one-step secant (OSS) method requires less storage and computation than the BFGS method [10]. This method does not store the entire Hessian matrix; it assumes that at each recurring, the former Hessian was the identity matrix. Each variable is calculated as in (8).

$$X=X+m*d_X \quad (8)$$

where d_X is the search direction, m is selected to minimize the performance and X weight and bias variables. In order for the iterations to be successful, the search direction is computed from the new gradient and the previous search direction as in (9).

$$d_X=-g_X+A_C*XCW+B_C*d_{g_X} \quad (9)$$

where g_X is the gradient, XCW is the change in the weights on the previous iteration, and d_{g_X} is the change in the gradient from the last iteration [17].

Trainbr is a training function that updates weight and bias values using Levenberg-Marquardt Optimization-Bayesian Regularization [10]. Each variable is calculated as follows according to Levenberg-Marquardt as in (1). Also this Bayesian Regularization takes place within the Levenberg-Marquardt algorithm [18]. *Trainbr* minimizes a combination of squared errors and weights, and then determines the correct combination and use the Jacobian for calculations. This process is called Bayesian regularization.

Trainr is a training function that updates weight and bias values learning rules with incremental updates after each presentation of an input. Inputs are submitted in random order. For each epoch, all training vectors (or sequences) are each submitted once in an alternative random order, with the network and weight and bias values updated thus after each singular submission [19].

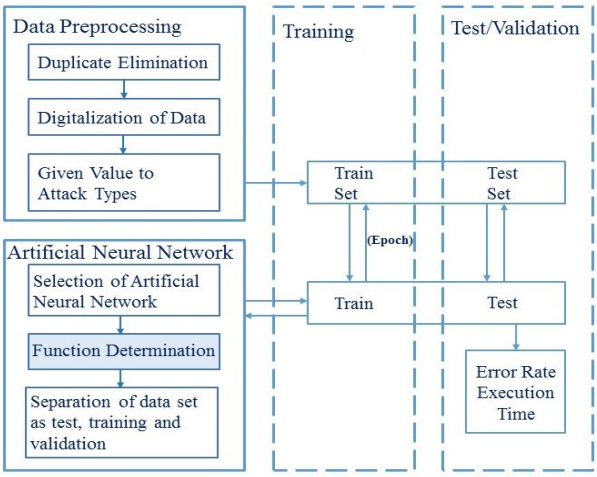

Fig. 2. Block Diagram of the proposed system

III. Proposed System

In this paper, it is aimed to implement an ANN based IDS system which has a multi-layer design. Therefore, in the first phase block diagram of the system is depicted as shown in Figure 2. Due to the fact that learning is a data mining approach, we used an accepted data set for implementing and testing our system. Firstly, the gathered data is processed before constructing and executing in the learning platform. Because there are some redundant data in it and also some data are stored in textual format, and for processing them we needed to convert them to numerical values.

After, preprocessing the data set, it is needed to construct the training set and testing set. After that the system is trained with the selected data in a number of times (this is automatically set as 100, in two of longer algorithms they are set as 10). In each iteration of the training, system is tested with the test-set to see the decreasing error rate of the system. The proposed NN system is implemented in 2-hidden layer structure as depicted in Figure 3.

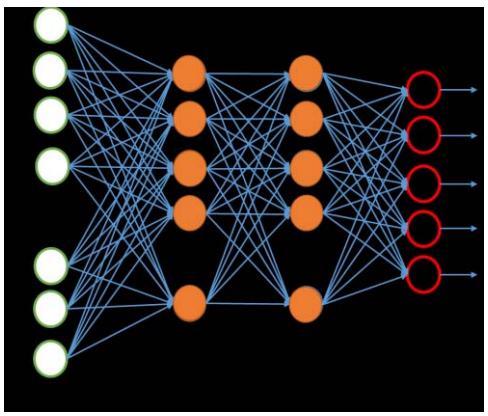

Fig. 3. 41*10*10*5 ANN Design of the proposed system with two level hidden layers

978-1-5386-3449-3/18/$31.00 ©2018 IEEE

For the acceptability of the proposed solution it is needed to select a globally accepted data set as detailed in the following subsection.

A. Dataset

KDD Cup'99 intrusion detection benchmark data set is used for the experiment of the proposed system [9]. This data set has a format as depicted in Figure 4.

As can be seen from this figure, there are 41 features which can be categorized as basic features, traffic features and content features, in the data set. Data contains some numeric information in binary and real number format; some text information about categories of the request. Additionally, this data set also contains one additional feature to show the label of the data whether it is an intrusion or not (by also depicting the intrusion type).

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattack.

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,110,3,1.00,1.00,0.00,0.00,0.03,0.06,0.00,255,3,0.01,0.06,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

Fig. 4. KDD Cup'99 Data Set Example

Therefore, before processing this data set in the neural network, there is a need of making a preprocessing as depicted in the block diagram of the proposed system (in Figure 3). In this processing the text data are converted to an integer format for being used in NN.

Size of the data set is also very important criteria. Mainly in standard data set of KDD Cup'99 there are about 5 million data. For testing the proposed system we selected a subset of them, then removed the redundant ones. The remaining data set size and properties are depicted in Table I.

TABLE I. DATA SET SIZE

Attack Type	*According to Attack Type*	*Total of all Attacks*	*Total*
Normal	47915	47915	77291
Dos	23570	29376	
Probe	2682		
R2L	3056		
U2R	68		

IV. EXPERIMENTAL RESULTS

To render networks more secure, IDSs tried to recognize intrusions with two major performance criteria: high detection and low false-alarm rates. Although they have some similarities in it, they also contain strict differences. In the study these parameters are organized in a single form and identified as error rate (misclassification rate) which is the reverse definition of accuracy of the system. This rate is calculated according to (10).

$$\text{Error Rate} = \frac{FP+FN}{TP+TN+FP+FN} \quad (10)$$

where TP means the true positive, TN means true negative, FP means false positive, and FN means the false negative rate of classification algorithms. Addition to them the execution of training time is also very important for taking into consideration. Although new technologies such as GPU and CUDA programming, the parallel training approach can decrease the execution time. However, in this paper we only focus on the serial execution on standard computer to compare the raw performance of the training functions.

We implemented our system with the MATLAB language by using previously implemented libraries and functions. Due to the advantage of matrix calculation the selected development platform, MATLAB is one of the mostly preferred environment for system developers. System is tested in two concepts: error rate and execution time. For making an appropriate comparison some platform properties are needed to be depicted as shown in Table II.

TABLE II. PLATFORM PROPERTIES

Properties	**Values**
Computer	Lenovo
CPU	Intel ® Core™ i5-6500 CPU @3.20GHz
Operating System	64 bit, Windows 10
RAM	8.00 GB (7.88 GB usable)
IDE	MATLAB R2016a Version

In this platform, we tested 7 different training functions with the same size of dataset. Due to the randomness of the selection we encountered with different performance and execution values. Therefore, all experiments are executed for 10 times and according to these experiments the maximum, minimum, average and standard deviation of the execution time are depicted in Table III.

TABLE III. EXECUTION TIME OF THE TRAINING FUNCTIONS

Function Type	*Min*	*Max*	*Average*	*Std.Dev.*
trainbfg	1199,5	1381,1	1281,3	63,7
trainbr	1145,8	1323,0	1223,9	61,6
trainc*	1323,0	3944,1	3678,0	219,4
traincgp	1382,7	1575,1	1471,6	67,9
trainlm	652,6	741,4	691,1	35,3
trainoss	1581,4	1789,6	1682,3	75,0
trainr*	2590,3	2965,6	2773,8	155,2
trainscg	**173,2**	**1664,6**	**474,3**	**626,0**

* Due to the high execution time these functions are tested 10 times, while others are 100

978-1-5386-3449-3/18/$31.00 ©2018 IEEE

Apart from this, error rates of the tested training functions are one of the most important performance criteria. Therefore, the maximum, minimum, average and standard deviation of the error rates of the training functions are depicted in Table IV.

TABLE IV. ERROR RATE OF THE TRAINING FUNCTIONS

Function Type	*Min*	*Max*	*Average*	*Std.Dev.*
trainbfg	6,51	32,00	12,19	5,21
trainbr	1,07	35,68	2,62	4,42
trainc*	5,87	8,41	6,67	0,72
traincgp	6,51	32,00	8,74	3,59
trainlm	**1,58**	**23,11**	**2,58**	**2,86**
trainoss	6,22	38,82	10,89	4,10
trainr*	2,50	3,59	2,83	0,38
trainscg	45,76	45,76	15,69	5,26

* Due to the high execution time these functions are tested 10 times, while others are 100

As can be seen from this table, each function has different convergence rates to their best values. Therefore, we also want to show these convergence rates in a single figure as depicted in Figure 5 (for 100 epoch samples).

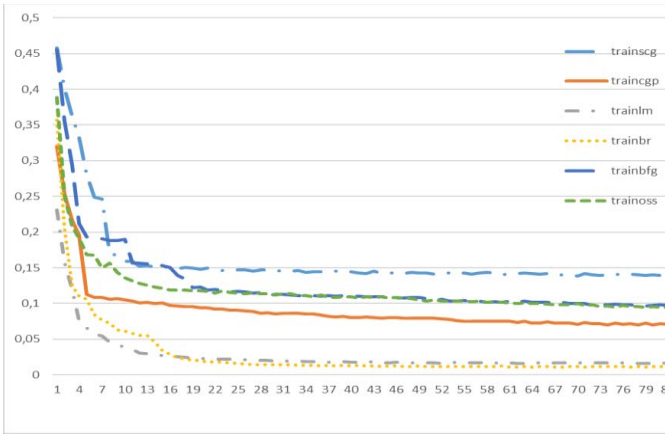

Fig. 5. Convergence rate of errors of the 100-tested trainin functions

Due to the high execution time of some training functions, 2 of them are repeated in 10 times. And their convergence rates are depicted in Figure 6.

As can be seen from these results, the best execution time is reached in the training function of *trainscg*. However, its performance on the correctness is not acceptable. If we look at the error rate of the tested function best solution is reached with the *trainlm* function and its execution time is relatively acceptable. At the same time if we look at the time consuming training functions we can see that they can produce an acceptable result in a smaller epoch number. Especially the result of *trainr* is very good even if it is tested only 10 iterations.

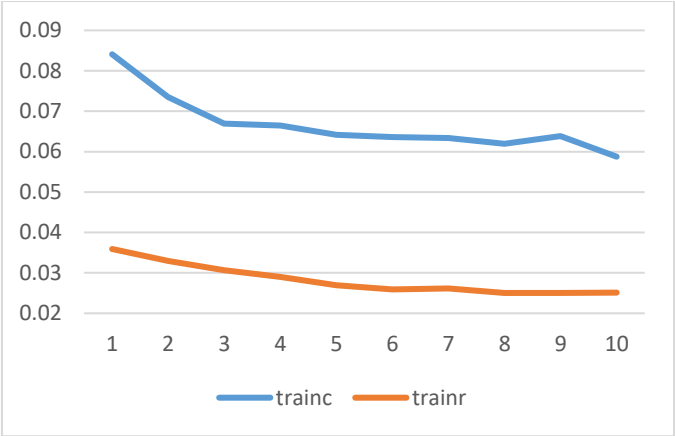

Fig. 6. 41*10*10*5 ANN Design of the proposed system with two level hidden layers

V. CONCLUSION

There are several training algorithms in artificial neural network approach. The performance of each algorithm mainly depends on the complexity and characteristic of the application area. In this paper, it is aimed to focus on the performance of selected training functions in application area of Intrusion Detection Systems. Therefore, firstly, a neural network-based intrusion detection system is implemented in a 2 hidden-layered design, and then the performances of the algorithms are compared with the highly preferred data set of KDD Cup'99.

The experimental results showed that *trainlm* function is the best algorithm on IDS application area which was implemented as a pattern recognition problem with five different patterns as: DoS, U2R, R2L, Probe and Normal. However, a time-consuming function of *trainr* also produces a very good result in a less number of epochs.

REFERENCES

[1] Worldwide Infrastructure Security Report, vol XI, ARBOR Networks, https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf, 2016.

[2] R.L. Adams, "Top Online Threats To Your Cybersecurity And How To Deal With Them", Forbes magazine, April 2017.

[3] U. Oktay and O. K. Sahingoz, "Proxy Network Intrusion Detection System for cloud computing," 2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), Konya, 2013, pp. 98-104.

[4] O. Can and O. K. Sahingoz, "A survey of intrusion detection systems in wireless sensor networks," 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), Istanbul, 2015, pp. 1-6.

[5] L. Van Efferen and A. M. T. Ali-Eldin, "A multi-layer perceptron approach for flow-based anomaly detection," 2017 International Symposium on Networks, Computers and Communications (ISNCC), Marrakech, 2017, pp. 1-6.

[6] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced Intrusion Detection System," 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, 2016, pp. 1-8.

978-1-5386-3449-3/18/$31.00 ©2018 IEEE

[7] R. Vinayakumar, K. P. Soman and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1222-1228.

[8] O. Can and O. K. Sahingoz, "An intrusion detection system based on neural network," 2015 23nd Signal Processing and Communications Applications Conference (SIU), Malatya, 2015, pp. 2302-2305.

[9] KDD Cup (1999) Intrusion detection data set. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[10] M.T. Hagan, H.B. Demuth and M.H. Beale, "Neural network design.", Vol. 20. Boston: Pws Pub., 1996.

[11] D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal on Applied Mathematics, Vol. 11, No. 2, June 1963, pp. 431–441.

[12] S. Roweis, Levenberg-marquardt optimization. Notes, University Of Toronto, 1996.

[13] R. Setiono, L.C.K. Hui, "Use of a quasi-Newton method in a feedforward neural network construction algorithm.", IEEE Transactions on Neural Networks, 1995, 6.1: 273-277.

[14] M.F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning. Neural networks", 1993, 6.4: 525-533.

[15] R. Fletcher and C.M. Reeves, "Function minimization by conjugate gradients," Computer Journal, Vol. 7, 1964, pp. 149–154.

[16] L.M. Saini and M. Kumar Soni. "Artificial neural network-based peak load forecasting using conjugate gradient methods." IEEE Transactions on Power Systems 17.3 (2002): 907-912.

[17] R. Battiti, "First and second order methods for learning: Between steepest descent and Newton's method," Neural Computation, Vol. 4, No. 2, 1992, pp. 141–166

[18] D.J. Mackay, "Bayesian interpolation.", Neural computation, 1992, 4.3: 415-447.

[19] H. Demuth, M. Beale, and M. Hagan, "Neural network toolbox™ 6. User's guide", 2008.

978-1-5386-3449-3/18/$31.00 ©2018 IEEE