Determining the Number of Hidden Neurons in a Multi Layer Feed Forward Neural Network

Lynn Ray
University of Maryland University College
3501 University Blvd East
Adelphi, MD 20783. USA

ABSTRACT: *A neural network intrusion detection system (IDS) can be effective against network attacks. However, their effectiveness can be reduced by changes in the neural network architecture. One problem is determining the number of hidden layer neurons. This can lead to reduced detection and high failure rates. This paper describes the affects of architecture on the performance of IDSs while finding a means to choose the proper architecture and number of hidden neurons. This method reduces the need for trial and error in determining the number of hidden layer neurons in a multi layer feed forward neural network IDS.*

Keywords: Multi Layer Feed Forward Neural Network, Detection Rate, Failure Rate

Received: 12 March 2013, Revised 17 April 2013, Revised 22 April 2013

1. Introduction

Hackers are using dynamic methods to attack network systems. Detection of these attacks requires the network to use a method that can learn and adjust its protection to prevent damage to computer systems. The best detection method to use involves recognizing abnormal behavior and taking action to stop an attack. Neural network IDS's utilize anomaly-based detection that can accomplish this task. [11] support this claim and found these types of networks work well in a dynamic environment. These devices are trained using normal and simulated attack data. Once trained, they can automatically recognize and block malicious traffic. However, these can be difficult to build and optimize due to using trial and error methods for determining the best architecture structure. This paper describes a method for determining the number of hidden neurons using their relationship to the number of input and output neurons. This relationship can help determine the optimized architecture for neural network IDSs without having to rely on trial and error. This paper looks at the most common architecture neural network IDS model composed of a multi-layer feed forward neural network (MLFFNN).

2. Multi layer feed forward neural network

The model used in this study was the common MLFFNN IDS. Its architecture is a commonly used model for working with anomaly-based IDSs [17, 4]. It works well in determining irregularities within large amounts of data such as those used in the publicly available NSL-KDD dataset [3]. This characteristic is very important when trying to find specific intrusions in today's global networks. These networks can collect large amounts of data in minutes that need to be analyzed in seconds to quickly determine if an attack is taking place. Neural networks are well suited to finding intrusions in large datasets.

Part of this model also uses algorithms such as the back propagation (BP) to help in detecting attack behavior. This algorithm is commonly used with neural networks and was selected in this study. The BP algorithm provides a means to compensate for errors collected in the output of the neural network. These errors are generated when the expected or target detection rate doesn't match the actual detection rate during each pass through the neural network IDS. With BP, the error is fed back into the neural network to adjust the connection weights and hopefully reduce the errors in the next pass through. The paper will concentrate on this model as depicted in figure 1.

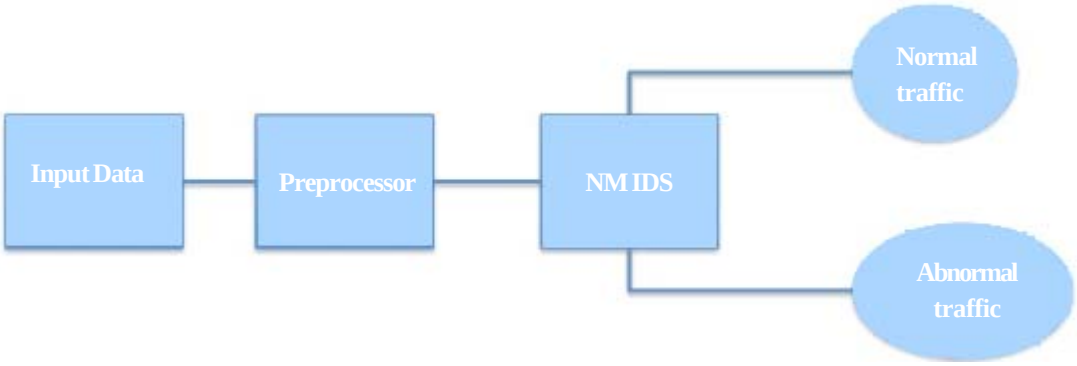

Figure 1. Neural Network IDS model

2.1 Input Data

In this model, the NSL-KDD dataset was used to provide test data for the research. This publicly available dataset has been used by researchers as well as the KDD 99 dataset. However, the KDD 99 dataset contains some problems. It has some duplication of items that could negatively affect or cause errors in the measurements. Also it has an imbalance of data across each of its four categories of attack types. A modification was done to resolve these problems by the introduction of the NSL-KDD dataset. It is smaller and has reduced the amount of data duplication and provided a better balance of data in each category. Thus, the NSL-KDD dataset was used.

2.2 Preprocessing

The NSL-KDD dataset still had a similar problem to the KDD 99 dataset. It contained string data such as TCP, IP, ICMP, Normal, Neptune, etc. A neural network can only process numeric data or it would generate an error. Thus, the dataset had to be converted to numerical data in order to process correctly through the IDS model. Preprocessing of this information is usually done by hand. An example of this conversion involved creating a conversion table for each type of string data. The protocol, flag and attack columns in the dataset had to be converted using this table. Each column has its own numbering system from 1 to 100. For example, the protocol used *tcp* =1, *icmp* = 2 and *ip* = 3. Once completed, the modified data can then be feed into the input layer of the neural network model.

2.3 Architecture

The neural network model is composed of three layers of neurons: input, hidden and output. Each neuron is constructed to act like the neurons in the human nervous system. Information propagates through the MLFFNN input layer then it forwarded to the hidden layer and passes into the output layer using a transfer function [17]. Figure 2 illustrates this action.

The input layer works to take in numeric traffic data that was preprocessed. The number of neurons in the input layer is determined by the number of intrusion features to be processed from the input data. The NSL-KDD dataset used in this experiment had 41 different features that equated to 41 input neurons. The hidden layer takes the output from the input layer and determines if the data is an intrusion or not. The number of hidden neurons is usually determined by trial and error and affects the detection rate of the IDS. Currently, there is no equation or other method for quickly determining the number of hidden neurons to use.

The result from the hidden layer is then sent to the output layer where it is classified according to the specific type of intrusion or as normal traffic. The number of output neurons is determined by the specific attacks being identified. The greater the number of specific attacks the larger number of output neurons. For example, if one wanted to just detect an attack but didn't care what

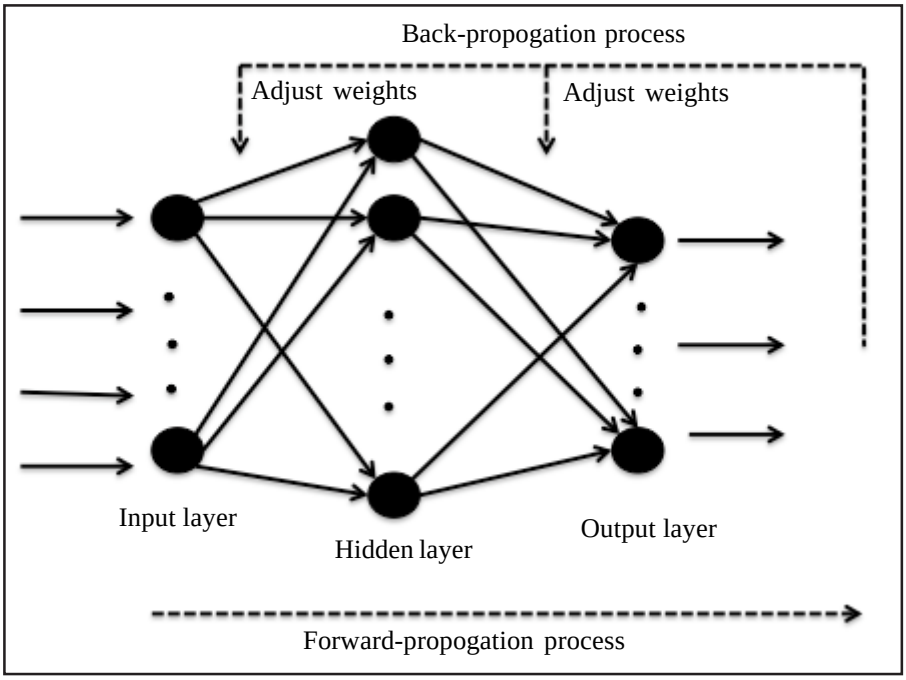

Figure 2. Feed forward neural network structure with BP algorithm

type, they could use just one neuron. This one output neuron could use an output of zero to mean normal traffic and one to identify an attack. For determining more specific attacks, one would use more output neurons based on the number of attack categories you are trying to identify. For detecting the four categories and normal traffic within the NSL-KDD dataset, one uses five output neurons. One neuron is used to identifying normal traffic and the other four for classifying different attack categories. Therefore, it is easy to determine the number of input and output neurons in the architecture of a multi layer feed forward neural network. This study will focus on defining the relationship of the number of hidden layer neurons to the number of input and output layer neurons. The end result will be to easily approximate the best number of hidden layer neurons without all the trail and error.

3. Related work

Calculating the neurons for the hidden layer can be difficult according to [17]. This is because too many hidden layer neurons and the training time increases. Too few and the detection rate drops. Determining the number of neurons in the hidden layer is done by trial and error [17]. This can be very time consuming and prone to errors. According to [7], it requires one to have experience to determine the number of hidden layer neurons. The reason for this problem is the lack of a mathematical equation to determine the number of hidden layer neurons [2]. So researchers have to rely on manually varying the number of neurons in the input, hidden and output layers until the best detection and failures rates are obtained. Researchers [16] and [17] and Aiguo (2010) each varied the number of hidden neurons in determining the optimum number. Some researchers used a relationship to the number of input neurons in determine the number of hidden layer neurons but these were very limited in scope. [11] used a relationship of n, $2n$ and $n/2$ where n is the number of input neurons. Our approach uses a wider range of variance over a fixed number of input neurons. This research attempted to define the relationship of these neurons to reduce the need for trial and error to calculate the best numbers. The study would find an optimal method of determining the number of hidden neurons by knowing the number of input and output neurons.

Varying the number of hidden neurons can affect the detection and failure rates. It is hard to get a proper number of neurons to produce a satisfactory value for detection and failure rates. Rastegari, Saripan and Rasid (2009) found that even a small variance of the number of hidden layer neurons can produce error in the training and testing of a neural network. This can lead to using a small amount of hidden layer neurons to avoid this issue [9]. Finding the optimal number of hidden layer neurons varies

and is not consistent between researchers. Increasing the number of neurons increases accuracy (performance) but decreases the convergence rate [22, 6]. [17] found that too few neurons cause more errors while too many result in poor convergence rates. Thus, its hard to determine an accurate number of neurons for the hidden layer [5]. In one model, varying the number of hidden layer neurons between 25 and 40 only caused a detection rate deviation of less than 4% [16]. However, no quantitative data was available to determine the affects on the detection rate for a number of hidden layer neurons above 40 or below 25. Therefore, it is a good practice to vary the number of hidden layer neurons to find the optimal topology [12], Stewart, Feng & Akl, 2010). The trial and error method of determining the number of neurons is too time consuming for optimizing an IDS model. A better method is needed. This research will determine an optimal relationship of hidden neurons to the input/output neurons while keeping the detection and failure rates at the best levels.

It has been stated that varying the number of input, hidden and output layer neurons to optimize a neural network IDS model can be difficult. Changing one variable at a time can also be time consuming [12]. Therefore, it is best to vary many parameters to find the optimal model. Our research used this approach to find the optimal IDS model. We decided to vary the number hidden and output layer neurons as well as the amount of data for training, validation and testing. However, using fixed amounts such as how Ahmad, [2] did in their research, may not provide an optimal solution. They used a fixed number of hidden layer neurons that was 3, 4, 6, 9, 10, 14, 15, 20, 25 and 30. In our research, we varied the number of hidden neurons to a greater extent that fine-tuned the number used. Thus our model would determine an optimal model with the best performance. [5] supports this idea.

4. Methods used

A MLFFNN IDS model was built using MATLAB R2013a 64-bit. MATLAB is commonly used to construct and evaluate algorithms and neural networks. The model ran on a 2.7 GHz Quad Core iMAC with 8GB RAM and OSx 10.8.2 Mountain Lion. To train and test the model, the NSL-KDD 20% training dataset was used. This dataset is a publicly available dataset used for training and testing neural network IDSs. It is composed of normal and four categories of attacks (Probe, Denial of Service, R2L and U2R). R2L pertains to someone attempting unauthorized access from a remote location. U2R involves unauthorized access to the root or administrator level of a system [15]. There are 41 features in the dataset that define each of the threat categories and one feature for defining when normal or attack traffic was found.

As described earlier, the NSL-KDD dataset was composed of numeric and symbolic data. However, neural networks can't process nonnumeric data so these values had to be converted [10, 13]. The symbolic data was converted to numeric to allow processing in the neural network. This involved creating a table to track the string values being converted and the numeric conversion. For example the protocol type was converted to *tcp* =1, *ip* = 2, *udp* = 3 and *icmp* = 4. A similar numbering conversion was done for the service, flags and target data types in the dataset. This was similar to the process followed by [8, 13]. Then the dataset was broken up into an input dataset of a matrix composed of the first 41 features and 25,192 rows. This composed the input data file used to training and test the MLFFNN IDS.

Two target files were composed from a portion of the NSL-KDD dataset based on the detection of single or multiple attack types. The single attack type target file could detect whether the information was normal or malicious using a column containing a binary one or zero. These target files were composed of 25,192 rows so as to match the input data file. The second target file used the same number of rows and had five columns defining the normal and four categories using a sequence of ones and zeros. The normal condition was defined as 1 0 0 0 0. The attacks were categorized as: *Probe* – 0 1 0 0 0, *DoS* – 0 0 1 0 0, *R2L* – 0 0 0 1 0, and *U2R* – 0 0 0 0 1. This was more detailed than the method used by Rastegari, Saripan and Rasid (2009) which used 3 digits. A large dataset was used to provide greater accuracy in determining the number of hidden neurons needed to optimize the detection and failure rates in the model.

The number of output neurons used was one and five to determine its effects on the detection/failure rates and number of hidden neurons. The number of hidden layer neurons was adjusted to find the optimal detection and failure rates based on a set number of hidden neurons. The number of hidden neurons was varied from 5 to 90 to help determine the relationship to the detection and failure rates. This helped determine the relationship of hidden neurons to both input and output neurons. Maximizing the number of input neurons and setting the number of output neurons to one and five helped to determine the best number of hidden neurons to optimize the model. Each test was repeated at least three times to ensure an adequate reading and

to compensate for errors. Only one variable was adjusted at a time. Thus, the output number of neurons was adjusted and the MLFF NN trained and tested. Then the number of hidden neurons was adjusted and the same tests repeated. Each time quantitative data was collected to measure differences in the detection and failure rates.

To further define the best model, the amount of training data used for validating and testing the model was varied from 5 to 20 percent of the total dataset used. This helped in determining the best amount of data to optimize the values of the detection/ failure rates and best value for the number of hidden neurons. The optimized value of hidden neurons was compared with the number of input and output neurons. This helped determine their relationship to each other. This relationship provided a means to determining a value for the number of hidden neurons in a MLFFNN.

5. Results

Varying the number of hidden layer neurons and output neurons from 1 and 5 resulted in some interesting findings. It was found that increasing the number of output neurons caused the training time or convergence rate to increase by a factor 2.2. This mans it takes more time to train a network to detect malicious activity. So if one wanted to detect different types of probes or Denial of Service attacks, the longer it takes to train a network to detect them. Also the detection rate dropped and failure rate increased when increasing the number of output neurons. This showed that trying to get more accurate attack information was time consuming and reduced the effectiveness of an IDS. Therefore, trying to determine specific attacks can have a negative effect on the performance of an IDS.

It was found that the number of hidden layer neurons had a varying effect on the detection rate (figure 3). There was about 0.5% variation in the detection rate over the different amounts of hidden layer neurons. The optimal number of hidden layer neurons

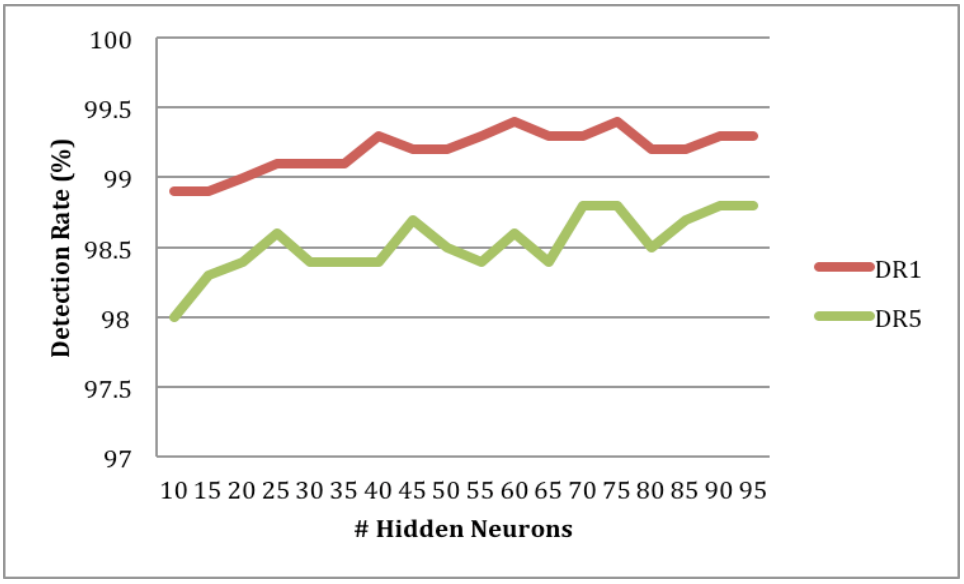

Figure 3. Number of hidden layer neurons to detection rate using 1 and 5 output neurons

was either 60 or 75. However, there was a very noticeable difference in detection rates when using a different number of output layer neurons. It seems that the best detection rate came when using a single output neuron (DR1) by at least a one percent difference when compared to 5 output neurons (DR5). After around 90 hidden layer neurons of just over 2x the number of input layer neurons, the detection rate dropped slightly and seemed to stabilize around 99.2%.

In comparison, the failure rates showed an improvement using the single output neuron as compared to the use of five neurons. This resulted in the single output neuron (FR1) having a lower failure rate of 0.5-1% compared to the 5 output neurons (FR5). The optimal failure rate was at 60 and 75 hidden layer neurons (figure 4). These were the lowest failure rates over the range of hidden layer neurons varied in the experiment.

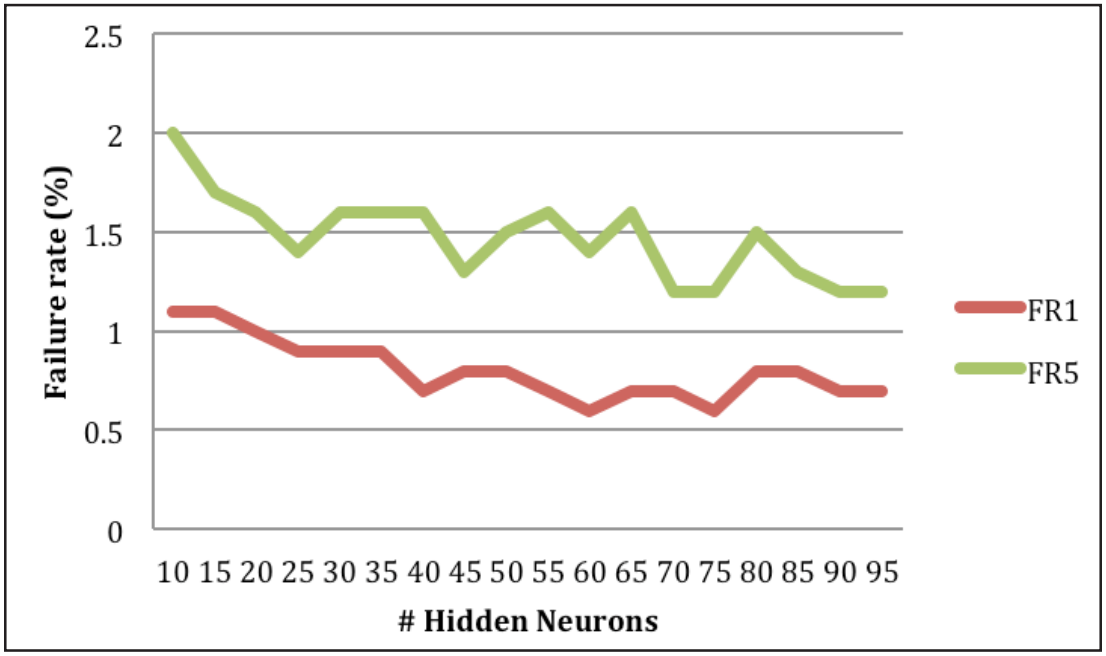

Figure 4. Number of hidden layer neurons to failure rate using 1 and 5 output neurons

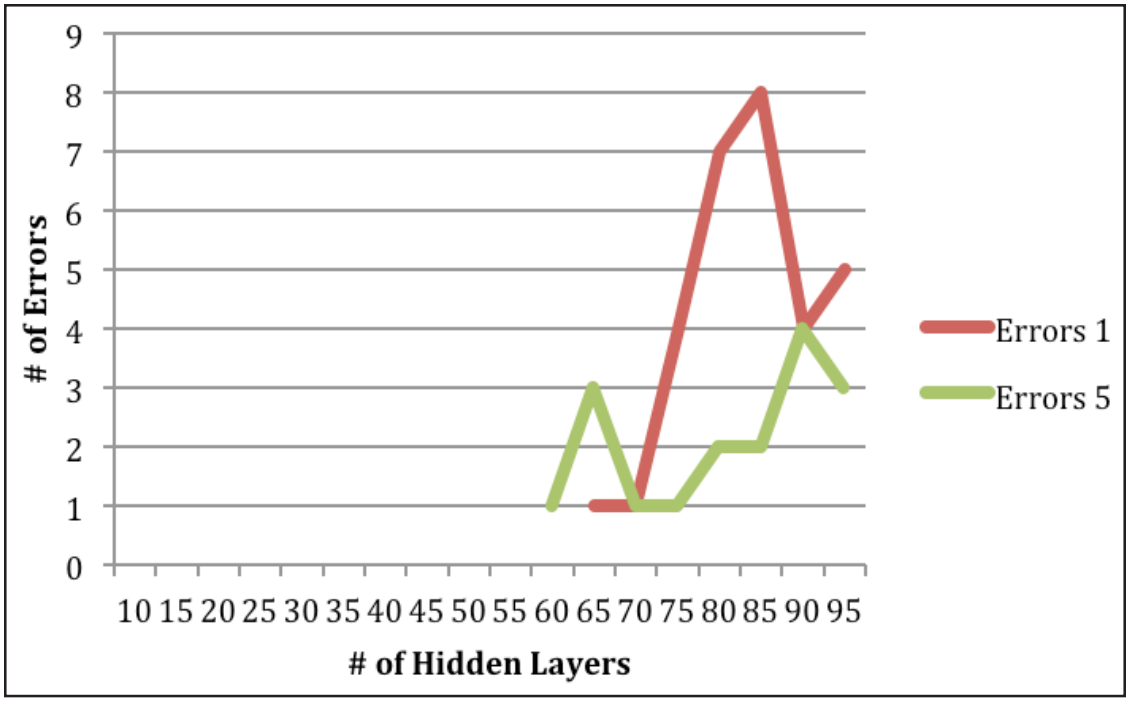

Figure 5. Number of errors found for single and 5 output neurons

Therefore, the best performance came from using 60 or 75 hidden layer neurons based on a single output neuron. However, it was found that at the higher number of hidden layer neurons the number errors or poor readings began to appear (figure 5). This had to be taken into consideration when determining the optimal number of hidden layer neurons because the errors gave a false reading that was unreliable. These errors started after the number of hidden neurons were above 60 and continued past 95. There seemed to be more errors found when using one output neuron (Errors 1) than 5 output neurons (Errors 5).

Taking all these conditions into affect, an optimal solution was found. It showed that using a single output layer neuron produced the best results for performance. Also the tests showed that determining the number of hidden neurons could be obtained using this relationship:

$$\#On < \# Hn < 2 \times \#In$$

Where *#On* is the number of output layer neurons, *#Hn* is the number of hidden layer and *#In* is the number of input layer neurons. Using this relationship, one can determine the number of hidden neurons without having to rely on time-consuming trial and error efforts. For example, take a neural network with 41 input neurons and five output neurons. The optimum number of hidden neurons found by trial and error was 55. Using the relationship equation, the value fell well within the $5 < \#Hn < 82$. To get a better approximation of the number of hidden neurons requires less input neurons and more output neurons. From this research, an equation for calculating the number of hidden neurons in a multi layer feed forward neural network is:

$$\#Hn = 1.34 \times \#In$$

To prove this equation, one uses the data from the neural network architecture and can come up with similar results. Thus, using 41 input neurons and one/five output neurons the optimal number of hidden neurons is 55. Now this equation and architecture relationship can be used to approximate the number of hidden layers in a multi layer feed forward neural network IDS. The study showed that this method could reduce the time necessary for optimizing the number of hidden neurons.

6. Conclusion

This study looked at how to overcome the necessary to do trial and error for finding the optimal number of hidden neurons in a multi layer feed forward neural network IDS. It showed that there is a specific relationship between the number of input and output neurons to the number of hidden neurons. Knowing this, one can use the number of input layer neurons to reasonability determine the optimal number of hidden layer neurons to create a MLFFNN IDS. Using this relationship an equation was found that can allow researchers to calculate the number of hidden neurons to use without doing lengthy trail and error calculations.

Research should not stop here. The equation can use further improvements such as fine-tuning the number of output layer neurons. Incrementing the number of output neurons by one and increasing the range can help fine-tune the results and the equation. Also the number of input layer neurons should be varied as well. Other types of neural network models need to be tested as well. This will provide a means for researchers to select the equation that fits the model they are using.

7. References

[1] Abdulla, S. M., Al-Dabagh, N. B., Zakaria, O. (2010). Identify features and parameters to devise an accurate intrusion detection system using artificial neural network. *World Academy of Science, Engineering and Technology*, 70, 627-631.

[2] Ahmad, I., Abdullah, A. B., Alghamdi, A. S. (2009). Application of artificial neural network in detection of DOS attacks. Presented at the *Second International Conference on Security of Information and Networks*, p. 229-234. ACM, October.

[3] Amer, S.H., Hamilton, J.A. (2009). Input data processing techniques in intrusion detection systems: Short review. *Global Journal of Computer Science and Technology*, 1 (2) 2-6.

[4] Bhaskar, T., Kamath, N., Moitra, S.D. (2008). A hybrid model for network security systems: Integrated intrusion detection system with survivability. *International Journal of Network Security*, 7 (2) 249-260.

[5] Bi, J., Zhang, K., Cheng, X. (2009). Intrusion detection based on RBF neural network. *In*: Proceedings of the 2009 International Symposium on Information Engineering and Electronic Commerce, p. 357-360. *IEEE Computer Society*, May.

[6] Bi, J., Zhang, K., Cheng, X. (2010). A new method of data processing in the intrusion detection system with neural network. Presented at the *Second International Workshop on Education Technology and Computer Science*, p. 343-345, IEEE Computer Society, March.

[7] Bouzida, Y., Cuppens, F. (2006). Neural networks vs. decision trees for intrusion detection. Presented at the *IEEE/1st Workshop on Monitoring, Attack Detection and Mitigation*, IEEE Computer Society, September.

[8] Choudhary, A. K. & Swarup, A. (2009). Neural network approach for intrusion detection. *In*: Proceedings of the 2nd *International Conference on interaction Sciences*, ACM. November.

[9] Gong, W., Fu, W., Cai, L. (2010). A neural network based intrusion detection data fusion model. Presented at the *Third International Joint Conference on Computational Science and Optimization*, p. 410-414, IEEE Computer Society, May.

[10] Ibrahim, L. M. (2010). Anomaly network intrusion detection system based on distributed time-delay neural network (DTDNN). *Journal of Engineering science and technology* 5 (4) 457-471.

[11] Joo, D., Hong, T., Han, I. (2003). The neural network models for IDS based on the asymmetric costs of false negative errors and false positive errors. *Expert Systems With Applications* 25. 69-75.

[12] Kilic, K., Bas, D., Boyaci, I.H. (2009). An easy approach for the selection of optimal neural network structure. *GIDA* 34 (2) 73-81.

[13] Moradi, M., Zulkernine, M. (2004). A neural network based system for intrusion detection and classification of attacks. *In*: Proceedings of the 2004 *IEEE International Conference on Advances in Intelligent Systems Theory and Applications*, IEEE Press, November.

[14] NSL-KDD. Retrieved from http://iscx.ca/NSL-KDD/

[15] Salimi, E., Arastouie, N. (2011). Backdoor detection system, using artificial neural network and genetic algorithm. Presented at the *International Conference on Computational and Information Sciences Conference*, p. 817-820, IEEE Computer Society, October .

[16] Wang, H., Ma, R. (2009). Optimization of neural networks for network intrusion detection. Presented at the *First International Workshop on Education Technology and Computer Science*, p. 418-420, IEEE Computer Society, March.

[17] Wei, Z., Hao-yu, W., Xu, Z., Yu-xin, Z., Ai-guo, W. (2010). Intrusive detection systems design based on BP neural network. Presented at the *Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, pages 462-465, IEEE Computer Society, August.