Comparative Study of Field Oriented Control of an Induction Motor Drives with PI Controller or with Sliding Mode Controller

Walid Emar, Maher Dababneh, Issam Ttrad
Isra University
Jordan
{walidemar, Issam_161}@yahoo.com, mikemaher__1@hotmail.co.uk

ABSTRACT: *Three phase induction motors are very widely used in industry that in some fields no other type of electric machines can be found. An a.c. motor drive connected to a load, directly or through a transmission gearbox, with the necessary control, supply and embedded systems such as microprocessors, power electronic converters, transformers and sensors constitute an electric drive system [1].*

The speed of an induction motor can be changed over a wide range by varying the supply voltage, frequency, slip, and poles. Changing the number of poles is a very known technique, and, if the motor has that possibility, it is then easily to go from one speed level into another by simply changing the position of the right switch [2-4].

In this paper, sliding mode controller is designed which results in good speed control and good performance for induction motion. Control schematic diagrams and units developed in this paper for the gain and band width of the controller are obtained for rotor resistance variation, model in accuracies and load disturbance, to have an ideal speed tracking. The chattering effect is also taken into consideration. The controller is simulated under various conditions and a comparative study of the results with that of PI controller has been presented. For sliding mode controller, Lyaponov stability method is applied to keep the nonlinear system under control. The sliding mode approach is method which transformed a higher-order system into first-order system. In that way, simple control algorithm can be applied, which is very straightforward and robust?

Keywords: Induction Motor Drive, PI Controller, Sliding Mode Controller, Chattering Effect

Received: 24 November 2012, Revised 27 December 2012, Accepted 12 January 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. [7] 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 trial and error. This paper looks at the most common architecture composed of multi layer feed forward neural network IDS's.

2. Multi layer feed forward neural network

The model used in this study was the common multi layer feed forward neural network IDS. This network is a feed forward network utilizing the back propagation algorithm. Its architecture is a commonly used model for working with anomaly-based [11] [4]. It works well in determining irregularities within large amounts of data [3]. This is very important when trying to find specific intrusions in today's global networks. Neural networks are well suited to finding intrusions in large datasets. The paper will concentrate on this model.

2.1 Preprocessing

Before data can be entered into the neural network model, it has to be converted to numerical data in order to process correctly through the IDS model. This is because the model uses only numerical data and raw network traffic data may contain alphanumerical information. Preprocessing of this information is usually done by hand. An example of this conversion involving the protocol such as *tcp* = 1, *icmp* = 2 and *ip* = 3. Once completed, the modified data is sent to the input layer of the neural network model.

2.2 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 MLFF NN input layer then forwarded to the hidden layer and passes into the output layer using a transfer function [11]. 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 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 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. 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. This way one can 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 [11]. Determining the number of neurons in the hidden layer is done by trial and error [11]. This is because there is a lack of mathematical equations to determine the number of hidden layer neurons [5]. Researchers [10]and [11] each varied the number of hidden neurons in determining the optimum number. This research found an optimal method of determining the number of hidden neurons without performing trial and error attempts.

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. 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 [1] [6]. [11] found that too few neurons cause more errors while too many result in poor convergence rates. Thus, its hard to get 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% [10]. 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 [8] (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.

4. Method used

A multi layer feed forward neural network IDS model was built using MATLAB x.x. The model ran on a 2.7 GHx Quad Core iMAC with 8GB RAM with OS x 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). There are 41 features in the dataset that define each of the threat categories and one feature for defining normal or attack found. It was composed of numeric and symbolic data. The symbolic data was converted to numeric to allow processing in the neural network. Then the dataset was broken up into an input dataset of a matrix composed of the first 41 features and 65,535 rows. Two target files were composed 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 one or zero. It was also composed of 65,535 rows. 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. 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 was varied between 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 optimize 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.

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 multi layer feed forward neural network.

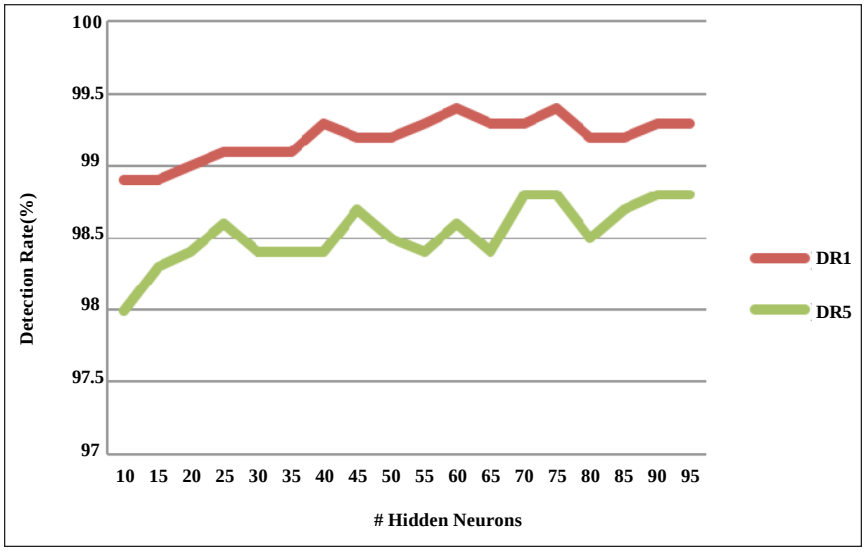

Figure 1. Number of hidden layer neurons to detection rate using 1 and 5 output neurons

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 between the different value of output neurons. Also the detection rate dropped and failure rate increased when increasing the number of output neurons. This showed there was a negative effect when trying to get a specific attack detected.

The number of hidden layer neurons had a negative effect on the detection rate (figure 1). It seems that the best detection rate

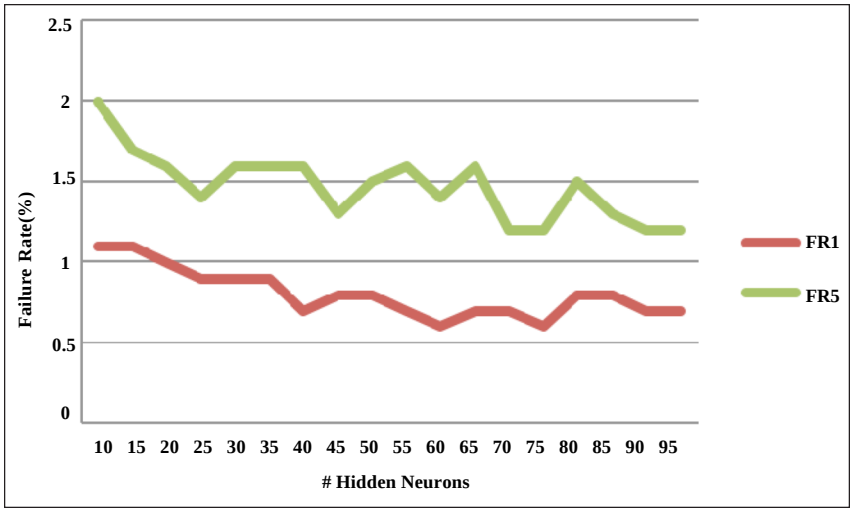

Figure 2. Number of hidden layer neurons to failure rate using 1 and 5 output neurons

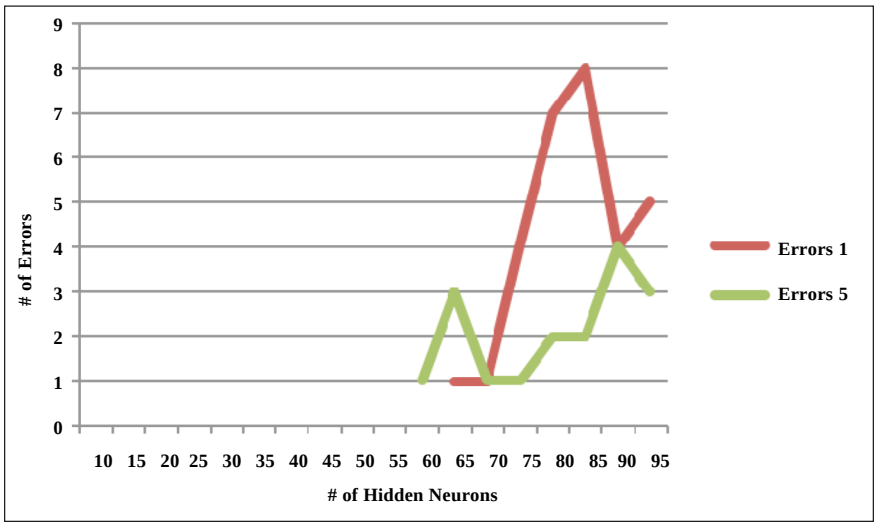

Figure 3. Number of errors found for single and 5 output neurons

came when using a single output neuron (DR1) by at least one percent difference when compared to 5 output neurons (DR5). The optimal number of hidden layer neurons was either 60 or 75. After these values the detection rate dropped slightly and seemed to stabilize around 99.2%.

In comparison, the failure rates showed between 0.5-1% in favor of the single output layer neuron (FR1) as compared to 5 output neurons (FR5). The optimal failure rate was at 60 and 75 hidden layer neurons (figure 2).

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 3). This had to be taken into consideration when determining the optimal number of hidden layer neurons. 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 can be found. It shows that the single output neurons produces the

best results for performance. Also the tests showed that determining the number of hidden neurons could be obtained using the relationship:

$$\#On < \# Hn < 2x \#In \quad (1)$$

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:

$$\#HN = 1.34 \; x \; \#input\ neurons \quad (2)$$

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 relationship between the number of input and output neurons to the number of hidden neurons. 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. However, further work is needed to test these against other models using different algorithms. Use of real traffic data should be used to test the relationships and equation.

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] 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.

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

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

[10] 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.

[11] 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*, p. 462-465. IEEE Computer Society, August.