Helsinki University of Technology
Espoo Finland

Torque Vector Control Using Neural Network Controller for Synchronous Reluctance Motor

X. Feng

Teco-Westinghouse Motor Co.
R&D Center
PO Box 0277, 5100 N. IH-35
Round Rock, TX 78628, USA
Tel: +1 512 218 7608
Fax: +1 512 218 7565
Email: feng.xueqing@ieee.org

R. Belmans & K. Hameyer

Katholieke Universiteit Leuven
Div. ELEN, Dept. ESAT
Kard. Mercierlaan 94
B-3001 Leuven-Heverlee, Belgium
Tel: +32 16 32 1020
Fax: +32 16 32 1985
Email: ronnie.belmans@esat.kuleuven.ac.be

ABSTRACT

This paper presents the torque vector control technique using a neural network controller for a synchronous reluctance motor. As the artificial neural network controller has the advantages of faster execution speed, harmonic ripple immunity and fault tolerance compared to a DSP-based controller, different multi-layer neural network controllers are designed and trained to produce a correct target vector when presented with the corresponding input vector. The trained result and calculated flops show that although the designed three layer controller with tansig, purelin and hard limit functions has more processing layers, the neuron number of each layer is less than that of other kinds of neural network controller, thus requiring less flops and yielding faster execution and response.

Keywords: neural network, field oriented control, variable speed drives.

1 INTRODUCTION

In this paper, a brief review of neural networks is introduced first. Then an existing torque vector control scheme is discussed. The control system consists of three units: flux observer, switching unit and speed controller. The three phase currents and voltages are chosen as inputs of the flux observer to achieve the torque produced by the rotor and the stator fluxes accounting for their position and magnitude. These signals are supplied to the switching unit and its outputs are the control of the voltage source inverter. As the artificial neural network controller has the advantages of faster execution speed, harmonic ripple immunity and fault tolerance compared to a DSP-based controller, different multi-layer neural networks are designed and compared. For an optional switching unit, two layer neural network controllers with log-sigmoid and perceptron neuron layers having a hard limit transfer function, are designed. Using the perceptron learning rule, the weights and biases of the neurons are trained to produce a correct target vector when presented with the corresponding input vector. The trained result shows that the input layer at least has 13 log-sigmoid neurons and output layer has 3 perceptron neurons. The sum-squared error reaches zero when epoch equals 1262 and trained weights and biases are obtained. Another two layer neural network with hypertangent-sigmoid neurons as input layer and perceptron neurons as output layer is designed. The input layer at least has 11 hypertangent-sigmoid neurons and output layer has 3 perceptron neurons. The sum-squared error of the network reaches zero with the epoch going to 1255. The corresponding and trained biases and weights are also obtained. The third designed and trained neural network controller is a three layer neural network with a back-propagation neuron layer as input layer, an elementary Woldraw-Hoff neuron layer as hidden layer and a perceptron neuron layer as the output layer. The result shows that the input layer at least has 5 hypertangent-sigmoid neurons, the hidden layer has 3 purelin neurons while the output layer has 3 perceptron neurons. The sum-squared error of the network reaches zero with the epoch is 2186. Their related weights and biases are also obtained. To verify the corresponding output signals of the designed and trained three types of neural network controllers, simulation block diagrams are designed and simulated digitally. The corresponding output control signals and related input signals are simulated and the results demonstrate that three kinds of the designed and trained neural network controllers are correct. Also, the calculated flops show that the designed three layer neural network controller requires less flops. This means that although the designed three layer controller with tansig, purelin and hard limit functions has more processing layers, the neuron number of each layer is less than that of other kinds of neural network controller, thus requiring less flops and yielding faster execution and response.

2 NEURAL NETWORKS

Building intelligent technical systems that can model human behavior has always been a key target. Therefore, it is not surprising that a technology such as neural networks has created great interest.

Multi-layer artificial neural networks have been used successfully for a wide variety of applications such as pattern recognition, image and speech processing, control and identification. It has been recognized that a controller based on multi-layer neural network architectures offers a promising way of handling complex control problems. Artificial neural networks are computing architectures inspired by the biological nerve system. They are useful in applications where formal analysis would be extremely difficult or even impossible. The behavior of a neural network is defined by its architecture, the way its individual computing elements are connected and the strength of those connections, or weights. The weights are adjusted by training the network according to a specified learning rule until it performs with the desired error rate.

Neural networks are information processing systems. In general, neural networks can be thought of as "black box" devices accepting inputs and producing output signals. Some of the operations that neural networks perform include:

- classification: an input pattern is passed to the network, and the network produces a representative class as output;
- pattern matching: an input pattern is passed to the network, and the network produces the corresponding output pattern;
- pattern completion: an incomplete pattern is passed to the network, and the network produces an output pattern that has the missing portions of the input pattern filled in;
- noise removal: a noise-corrupted input pattern is presented to the network, and this removes some (or all) of the noise and produces a cleaner version of the input pattern as output;
- optimization: an input pattern representing the initial values for a specific optimization problem is presented to the network, and the network produces a set of variables that represents a solution of the problem;
- control: an input pattern represents the current state of a controller and the desired response for the controller and the output is the proper command sequence creating the desired response.

Neural networks consist of processing elements and weighted connections (Fig. 1). Each layer in a neural network has a collection of processing elements (PEs). Each PE in a neural collects the values from all of its input connections, performs a predefined mathematical operation called transfer functions (typically a dot product followed by a PE transfer function), and produces a single output value. The neural network in Fig. 1 has three layers: the transfer functions F_X, F_Y and F_Z consisting of the PEs $\{x_1, x_2, \ldots, x_m\}$, $\{y_1, y_2, \ldots, y_n\}$ and $\{z_1, z_2, \ldots, z_o\}$ respectively from left to right. The PEs are connected using weighted connections. In Fig. 1 there is a weighted connection from every PE F_X to every PE F_Y, as there is in the subsequent layer. Each weighted connection (often synonymously referred to as either a connection or a weight) both acts as a label and a value. The connection from x_1 to y_2 is called weight w_{12}. The connection weights are storing the information. The value of the connection weights is often determined by a neural network learning procedure, but sometimes are predefined and hardwired into the network. It is through the adjustment of the connection weights that the neural network is able to learn. By performing the update operations for each of the PEs, the neural network is able to recall information.

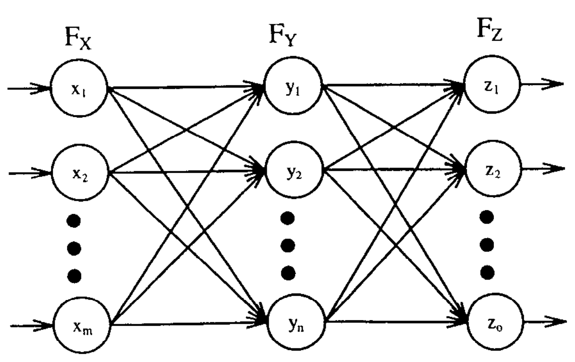


Fig. 1. Three layer neural network

There are several important features illustrated by the neural network of Fig. 1 that apply to all neural networks.

- Each PE acts independently of all others. Each PE's output relies only on its continuously available inputs from the abutting connections.
- Each PE relies only on local information. The information provided by the adjoining connections is all a PE needs to process. It does not require the knowledge of the status of any of the other PE's, not having an explicit connection.
- The large number of connections provides a large amount of redundancy and facilitates a distributed representation.

The first two features allow neural networks to operate efficiently in parallel. The last feature provides neural networks with inherent fault-tolerance and generalization quantities that are very difficult to obtain by other computing systems. Furthermore, neural networks are able to learn arbitrary non-linear mappings. This is obtained through proper arrangement of the neural networks, introduction of a non-linearity in the processing elements (i.e., adding a non-linear PE function), and using the appropriate learning rules. This is a powerful attribute.

There are three types of applications where neural networks are particularly advantageous.

- applications only requiring a few decisions from a massive amount of data (e.g., speech and image processing);
- applications where non-linear mappings must be automatically acquired (e.g., loan evaluations and robotic control);
- applications looking for a near-optimal solution to a combinatorial optimization problem rapidly (e.g., airline scheduling and telecommunication message routing).

In a broad sense, neural networks consist of three principle elements: topology, learning and recall. Topology means how a neural network is organized in layers and how these layers are connected. Learning explores how a neural network is trained and related information is stored. Recall shows how the stored information is retrieved from the network.

PE transfer functions, also referred to as activation functions or squashing functions, map a PE's (theoretically) infinite domain to a pre-specified range. Although the number of PE transfer functions possible is infinite, five are regularly employed by a majority of neural networks: linear, step, ramp, sigmoid and Gaussian.

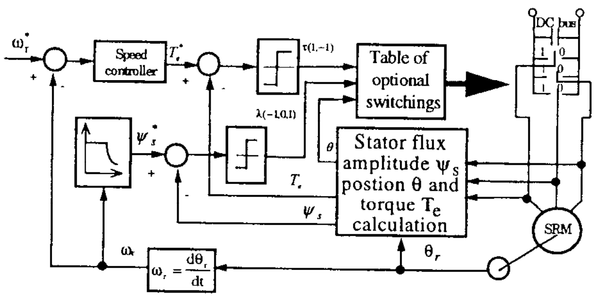
The most appealing quality of a neural network is its ability to learn. Learning, is defined as change in connection weight values resulting in the capture of information that can be recalled later. Several procedures for changing the values of connection weights, i.e. learning methods are developed: Hennian Correlations, Principal Component, Differential Hebbian, Competitive, Min-Max Classifier, Error Correction, Reinforcement and Stochastic Learning.

The wide range of learning procedures emphasizes the analysis and storage of information on target pattern. The emphasis of the recall techniques is on retrieving information already stored in the neural network. The feed-forward and feedback recall are two broad categories.

The principle topologies, learning algorithms and recall dynamics are the main elements of neural networks.

3 TORQUE VECTOR CONTROL OF A SYNCHRONBOUS RELUCTANCE MOTOR

The torque vector control scheme of a synchronous reluctance motor is shown in Fig. 2. The main units are the speed controller, the look-up table for switching, the stator flux amplitude, position and torque calculations and the voltage-source inverter. The torque vector control is based on the estimation of the torque and the amplitude and position of the stator flux. The three-phase currents and voltages are chosen as inputs of the stator flux magnitude, position and torque calculation unit. The difference between speed reference and actual speed is fed into a speed controller and the output is the value of the torque reference. The error of the torque reference and torque is supplied to a sign function unit to obtain the torque control signal *τ (-1, +1)*. The difference between stator flux and stator flux reference is used to produce a stator flux signal *λ (-1, 0, +1)*. The torque control signal, stator flux signal and stator position signal are transferred to a look-up table for switching, and its outputs are the control signals of the voltage source inverter linking the dc bus and the motor.



ω_r^*
Speed controller
T_e^*
τ(1,-1)
Table of optional switchings
λ(-1,0,1)
ψ_s^*
θ
T_e
ψ_s
Stator flux amplitude ψ_s postion θ and torque T_e calculation
θ_r
ω_r
$\omega_r = \frac{d\theta_r}{dt}$
DC bus
SRM

Fig. 2. Torque vector control of a synchronous reluctance motor

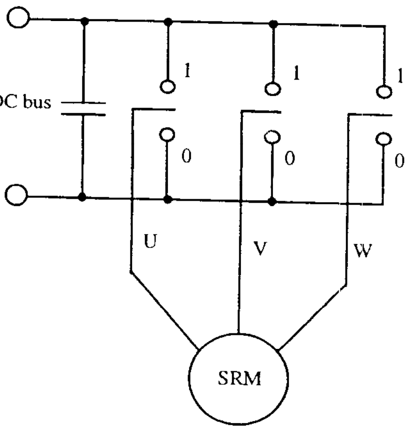
4 NEURAL NETWORK CONTROLLER FOR OPTIONAL SWITCHINGS

In the torque vector control of synchronous reluctance motor the optional switching pattern is used for connecting the three phase windings to the d.c. bus in order to produce the corresponding six voltages u_1, u_2, u_3, u_4, u_5, and u_6 (Fig. 3). The input signals of the look-up table for the optional switching are the stator flux sign (*λ=-1, 0, +1*), torque error sign (*τ=-1, +1*) and the stator flux positions (*θ*). Its output singles is as shown in table 1 [1].

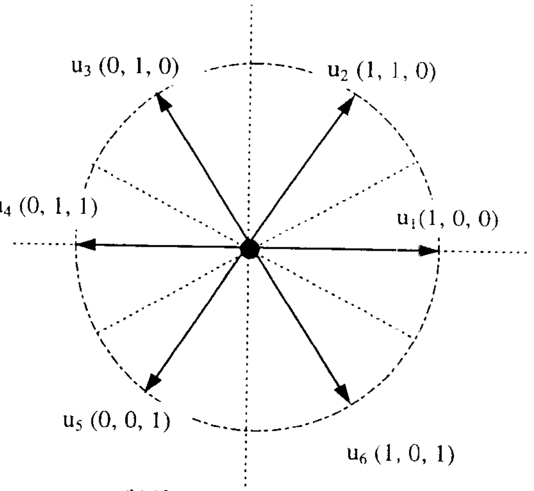
In the following, the focus is on the design of the two or three layer artificial neural network controllers.

Table 1 (the values of u_1~u_6 are as shown in Fig. 3b)

λ=+1	τ=+1	u_2	u_3	u_3	u_4	u_4	u_5	u_5	u_6	u_6	u_1	u_1	u_2
	τ=-1	u_6	u_1	u_1	u_2	u_2	u_3	u_3	u_4	u_4	u_5	u_5	u_6
λ=0	τ=+1	u_3	u_4	u_4	u_5	u_5	u_6	u_6	u_1	u_1	u_2	u_2	u_3
	τ=-1	u_5	u_6	u_6	u_1	u_1	u_2	u_2	u_3	u_3	u_4	u_4	u_5
λ=-1	τ=+1	u_2	u_3	u_3	u_4	u_4	u_5	u_5	u_6	u_6	u_1	u_1	u_2
	τ=-1	u_6	u_1	u_1	u_2	u_2	u_3	u_3	u_4	u_4	u_5	u_5	u_6
Stator flux position (θ)		0~ 30°	30°~ 60°	60°~ 90°	90°~ 120°	120°~ 150°	150°~ 180°	180°~ 210°	210°~ 240°	240°~ 270°	270°~ 300°	300°~ 330°	330°~ 360°



(a) Six pulse PWM inverter



(b) Non-zero voltage vector

Fig. 3. Principle of torque vector control

4.1 Perceptron neuron layer

The perceptron neuron layer, having a hard limit transfer function, is trained by the so-called perceptron learning rule. Each external input vector is weighted using an appropriate weight *w*. The sum of the weighted inputs is sent to the transfer function, which also has an input of *1* transmitted to it through the bias *b*. The transfer function, hardlim, returns a *0* or *1*. If the bias *b* is not used, the perceptron neuron produces a *1* if its net input is larger than *0*, otherwise a *0*. If a bias *b* is present, the function is shifted to the left by an amount *b*. The hard limit transfer function gives a perceptron the ability to classify an input vector by dividing the input space into two regions. Output vectors are either *0* or *1*, depending on the classification of the input. The output of the switching unit is either *1* or *0*, corresponding to switch turn-on or turn-off respectively.

4.2 Neural network controller with log-sigmoid and perceptron neuron layers

The logistic sigmoid, or log-sigmoid, activation transfer function is often used with neurons being trained using the back-propagation learning rule. It is used to map a neuron input from interval $(-\infty, +\infty)$ into the interval *(0, +1)*.

Back-propagation was created by generalizing the Widrow-Hoff learning rule to multiple layer networks and non-linear differentiable transfer functions. Input vectors and corresponding output vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors, or classify input vectors in an appropriate way. The back-propagation learning rules are used to adjust the weights and biases of the networks in order to minimize the sum squared error of the network. This is done by continuously changing the values of the network weights and biases in the direction of steepest descent with respect to the error. Changes in each weight and bias are proportional to the effect on that element on the sum-squared error of the network. To train a network, vectors are presented and the output and error vectors are calculated. The sum of the squared errors is evaluated. If the sum squared error for all training vectors is less than the error goal, training stops.

A neural network with log-sigmoid neurons as input layer and perceptron neurons as output layer is shown in Fig. 4.

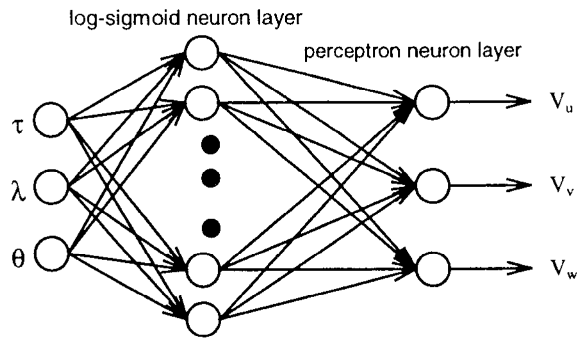


Fig. 4. Neural network with log-sigmoid and perceptron neurons

The input layer at least has *13* log-sigmoid neurons and output layer has *3* perceptron neurons. Two layer neurons have been trained and the sum-squared error of the network with the epoch is shown in Fig. 5. The sum-squared error reaches zero when epoch equals *1262*.

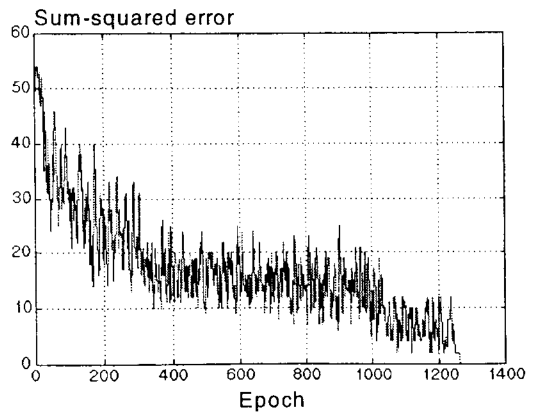


Fig. 5. Sum-squared error with epoch

4.3 Neural network controller with hypertangent sigmoid and perceptron neuron layers

A neural network with hypertangent-sigmoid neurons as input and perceptron neurons as output layer is shown in Fig. 6.

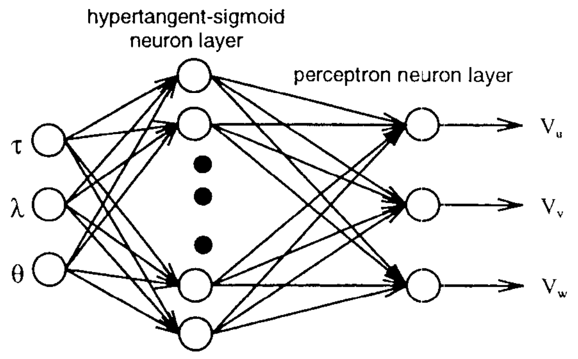


Fig. 6. Neural network with hypertangent-sigmoid and perceptron neurons

The input layer at least has *11* hypertangent-sigmoid neurons and output layer has *3* perceptron neurons. Two layer neurons have been trained and the sum-squared error of the network with the epoch is obtained. As shown in Fig. 7 the sum-squared error reaches zero when epoch equals *1255*.

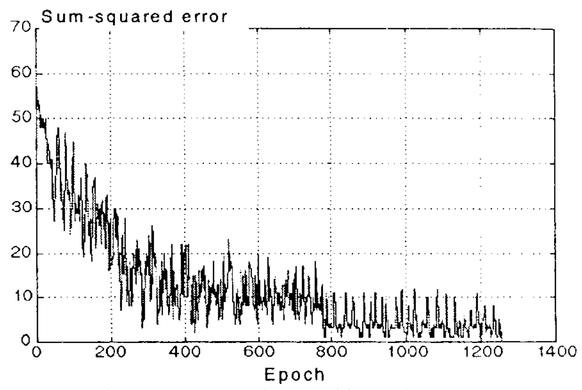


Fig. 7. Sum-squared error with epoch

4.4 Neural network controller with hypertangent sigmoid, purelin and perceptron neuron layers

A three layer neural network controller with hypertangent-sigmoid neurons as input, purelin neurons as hidden and perceptron neurons as output layer is shown in Fig.8.

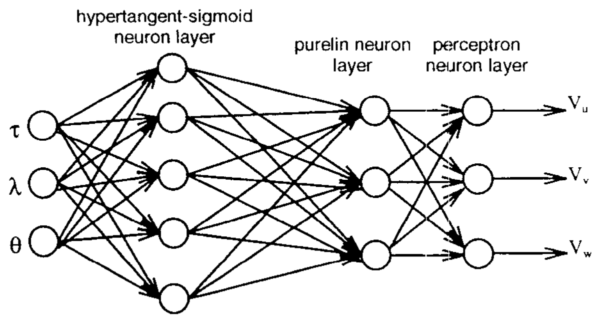


Fig. 8. Neural network with hypertangent-sigmoid, purelin and perceptron neurons

The input layer has at least *5* hypertangent-sigmoid neurons, the hidden layer has *3* purelin neurons while the output layer has *3* perceptron neurons. Three layer neurons have been trained and the sum-squared error of the network with the epoch is shown in Fig. 9. As shown in Fig. 9, the sum-squared error reaches zero when epoch equals *2186*.

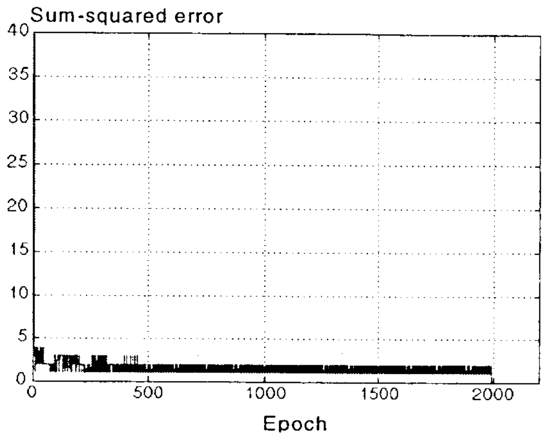


Fig. 9. Sum-squared error with epoch

5 SIMULATION AND COMPARISON FOR THREE NEURAL NETWORK CONTROLLERS

To verify the corresponding output signals of the designed and trained neural network controller, digital simulation techniques are used. For log sigmoid-perceptron, the hypertangent sigmoid-perceptron and the hypertanget-purelin-perceptron neural network controllers, the same corresponding output control signals related input signals can be simulated as shown in Fig. 10. The output control signals coincide with the look-up table of switching control indicated in the Table 1. The simulation results demonstrate that all three designed and trained neural network controllers are correct and can be used to replace the existing switching unit for the torque vector control scheme for a synchronous reluctance motor.

When designing a neural network architecture for a particular problem, it may be interesting to know how the number of floating point operations (flops) varies with the parameters of the network such as the layer number, transfer functions and neuron number. Less flops means faster execution and response of the neural network. Table 2 gives the calculation effort of the three kinds of the neural network controllers.

Table 2: Comparison of flops for three kinds of neural network controllers

Neural network	Logsig +Perceptron	Tansig +Perceptron	Tansig+Purelin +Perceptron
Flops	117+78	121+66	55+30+18

The flops show that the three layer neural network controller with hypertangent sigmoid (tansig) layer as input, pure linear (purelin) layer as hidden neuron and perceptron layer as output requires less flops. Although the three layer controller with tansig, purelin and hard limit functions has more processing layers, the neuron number of each layer is less than that of other kinds of neural network controller, thus requiring less flops and yielding faster execution and response.

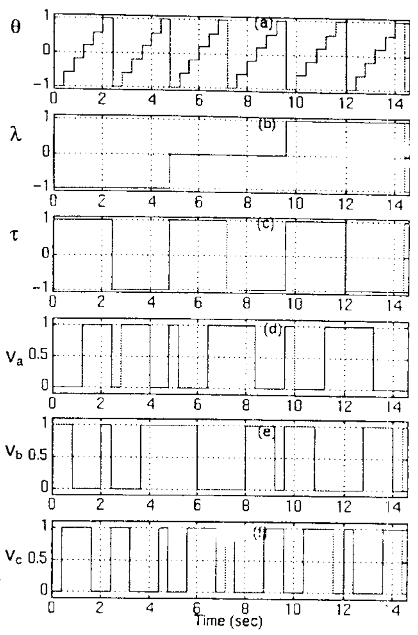


Fig. 10. Input signals and responses of neural network controller

6 CONCLUSIONS

For torque vector control with an artificial neural network a controller with one input and one output layer has been designed and simulated to replace the optional switching unit. Using different training methods such as Perceptron, Back-propagation and different learning rules, three kinds of the neural network are designed and trained to produce correct target control signals when presented with the corresponding input signals. The weights and bias for different neuron layers are obtained. The floating point calculations (flops) show that a three layer neural network with hypertangent sigmoid (tansig) transfer function as input layer, pure linear (purelin) transfer function as hidden layer and hard limit (hardlim) transfer function as output layer requires less flops. The three designed neural network controllers are digitally simulated. The results show that the output signals of the neural network controllers have a correct output compared to traditional optional switching unit, i.e., the designed neural network controllers are correct. The research on other control units with neural network controllers is on the way and results will be published.

REFERENCES

[1] I. BOLDEA, Z. X. FU and S. A. NASAR. "Torque Vector Control (TVC) of Axially-Laminated Anisotropic (ALA) Rotor Reluctance Synchronous Motors". *Electrical machines and Power Systems*, vol. 19, 1991, pp. 381-398.