ISSN: 2229-6956(ONLINE)
DOI: 10.21917/ijsc.2012.0048

NONLINEAR SYSTEM MODELING USING SINGLE NEURON CASCADED NEURAL NETWORK FOR REAL-TIME APPLICATIONS

S. Himavathi[1], A. Muthuramalingam[2], A. Venkadesan[3] and K. Sedhuraman[4]
Department of Electrical and Electronics Engineering, Pondicherry Engineering College, India
E-mail: [1]himavathi@pec.edu, [2]amrlingam@pec.edu, [3]a_venkyeee@pec.edu and [4]sedhu_k@pec.edu

Abstract
Neural Networks (NN) have proved its efficacy for nonlinear system modeling. NN based controllers and estimators for nonlinear systems provide promising alternatives to the conventional counterpart. However, NN models have to meet the stringent requirements on execution time for its effective use in real time applications. This requires the NN model to be structurally compact and computationally less complex. In this paper a parametric method of analysis is adopted to determine the compact and faster NN model among various neural network architectures. This work proves through analysis and examples that the Single Neuron Cascaded (SNC) architecture is distinct in providing compact and simpler models requiring lower execution time. The unique structural growth of SNC architecture enables automation in design. The SNC Network is shown to combine the advantages of both single and multilayer neural network architectures. Extensive analysis on selected architectures and their models for four benchmark nonlinear theoretical plants and a practical application are tested. A performance comparison of the NN models is presented to demonstrate the superiority of the single neuron cascaded architecture for online real time applications.

Keywords:
Single Neuron Cascading, Neural Networks, Modeling, Compact Models, Real-time Applications

1. INTRODUCTION

System modeling and identification is an area of active research as it forms the basis for efficient analysis and effective control. A good model should adequately describe the behavior of the system. It is desirable to have automation in design and ease in implementation. If complete knowledge of the system exists then mathematical models can be built from fundamental principles. A number of classical and statistical methods are available in the literature for mathematical modeling of systems. When no/partial knowledge of the system is available, models are built from input/output data. The data is used to obtain/derive the closest approximation to the actual underlying relationship between a finite set of input–output data.

Neural Networks have been widely used as nonlinear function approximators [1]-[3]. The performance of the Neural Network to a large extent depends on the network architecture, activation functions and learning algorithms. The network architecture, which is the method of interconnection between the neurons, determines the capability of a network to approximate a given function. Different neural architectures are proposed in literature. The popular architectures are single/multilayer perceptrons, Radial Basis Function networks (RBF) and cascade networks. The single/multilayer feed forward architecture uses inner product whereas the Radial Basis Function networks (RBF) use Euclidean distance. Hence the multilayer networks are more suitable for function approximation whereas RBF networks are more suitable for classification tasks. An attempt to use new activation functions which is a composite set of sigmoidal functions in RBF networks for better function approximation is reported in [4]. However as RBF is a single layer structure, and compactness similar to multilayer structures is difficult to achieve. The cascade multilayer architecture has been used in a number of pattern classification tasks as cascade correlation networks [5]. The cascade architecture differs from the multilayer Feed Forward (FF) architecture in the sense that neurons in a layer obtain inputs from all the previous layers unlike the FF architecture where only the previous layer outputs are fed as inputs. Different neural architectures and learning algorithms are currently under investigation and many related findings are reported in the literature [4]-[10]. Different activation functions are proposed to suit specific applications [4], [7], [8], [11]. For function approximation the tan-sigmoid function is widely used in multilayer networks and Gaussian functions are used in RBF networks. The learning algorithm determines the speed of learning and the optimization of the parameters of the network. Directed search methods such as gradient descent techniques [12] and random search soft computing techniques are used [13].

The structural compactness and computational complexity of the model assumes importance when used in online applications such as control and estimation. Related works to achieve compactness are reported in [4], [14]. The principle objective of this paper is to identify the most suitable NN architecture for such applications. The model has to be easy to design, simple to implement and fast to respond with good accuracy. Automation in design is reported in [13], [14].The parametric method of analysis is used to identify a neural network architecture which would be most useful to build simple and compact models.

Fahlman S.E. et al proposed multilayer cascade architecture with one neuron per layer and training was carried out using the cascade correlation learning technique for pattern classification [5]. Both the learning methods entail only training a subset of weights, with the remaining weights being frozen. There are merits and demerits in weight freezing method. The advantage of weight freezing is that there are far fewer weights to optimize than when all the weights are trained. The drawback is that weight freezing results in larger network. Constructive Cascade Network proposed by Nicholas K. L. et al [15] does not perform weight freezing. Since all weights continue to learn, smaller networks are often constructed.

Cascading single neuron in every hidden layer results the "Single Neuron Cascaded" (SNC) architecture. Such a network can be allowed to self grow till the target performance is reached and this allows automation in design of neural network model. Although the above design automation is feasible for single layer FF network, multilayer network design is difficult to automate

and optimize as it lacks the uniqueness in determining the number of layers and the neurons in each layer.

This paper identifies the SNC architecture to be most suitable for obtaining simple compact models for nonlinear function approximation tasks. A number of benchmark examples are considered to determine the effectiveness of such models using SNC-NN. The performance is compared with the more popular feed forward single/multi layer architecture based NN models. Extensive simulation studies including a practical example are carried out. The implementation of NN models using ADSP/FPGA and their performance are presented and discussed to draw the major conclusions of the present investigations on SNC-NN.

The paper is organized as follows. The section 2 presents the identified architecture and its suitability for function approximation and compares it with existing standard single/multilayer feedforward networks. Section 3 presents the simulation results of four standard benchmark examples and discusses the results obtained for each example. Section 4 presents a practical example realized trough SNC-NN model using ADSP/FPGA. Section 5 concludes the paper.

2. ANALYSIS OF SELECTED NEURAL ARCHITECTURES AND LEARNING ALGORITHMS

The observations on the performance of various NN models and their complexity of implementation for more demanding real time applications motivated the proposed parametric and performance analysis of neural network architectures. The identified Single Neuron Cascaded Architecture Neural Network (SNC-NN) model is compared with the more popular single/multilayer layer feed forward neural network. The parametric analysis is undertaken to highlight the characteristics of the architectures that are understood to be primarily responsible and important to achieve simple and compact NN models.

2.1 ARCHITECTURES

The architectures considered in the paper are the single neuron cascaded architecture and feed forward architecture.

2.1.1 Single Neuron Cascaded Architecture:

The Cascade architecture consists of an input layer, hidden layers and an output layer. The first hidden layer receives only external signals as inputs. Other layers receive external inputs and outputs from all previous (M-1) 1ayers/neurons. It is called cascade because the input to a neuron consists of system inputs and outputs of all preceding layers/neurons. This is in contrast to the feed-forward architecture where inputs to a neuron are only from previous layer. A cascade NN can have any number of neurons in each layer and the cascading of inputs improves its mapping capability. In this paper a single neuron is used in every hidden layer so as to obtain a single neuron cascaded architecture which is compact, self organizing and inherits the advantages of cascaded the inputs.

The Single Neuron Cascaded (SNC) architecture with multiple inputs/single output is shown in Fig.1. The number of inputs to a neuron is observed to increase proportionally with the number of layers. Each neuron in the architecture includes weights, bias and a nonlinear activation function. The weights of interconnections to the previous layer are called as "input weights" and the weights of interconnections between the layers are called "link weights". The tan-sigmoid activation function is used for all hidden layers while pure-linear function is used for output layer. Initially, a hidden layer with only one neuron between the input and output is trained. To create a multilayer structure, hidden layers are added one by one and the whole network trained repeatedly using the concept of moving weights so as to obtain more compact networks. This process continues, till the performance index is reached.

The weights and biases of all neurons are the parameters of the network and shown in Fig.1. The generalized formula to compute the total number of parameters of a given Cascade NN is presented. The cascade NN can have any number of neurons in each layer. The total number of parameters (P_C) for a cascade neural network is presented in Eq.(1). The first and second terms in Eq.(1) are deduced considering separately the weights and biases respectively. All prefixes denote layers and suffixes denote neurons in a layer. The proposed parametric method of analysis is effective to envisage the capability of cascade architecture in modeling highly nonlinear systems.

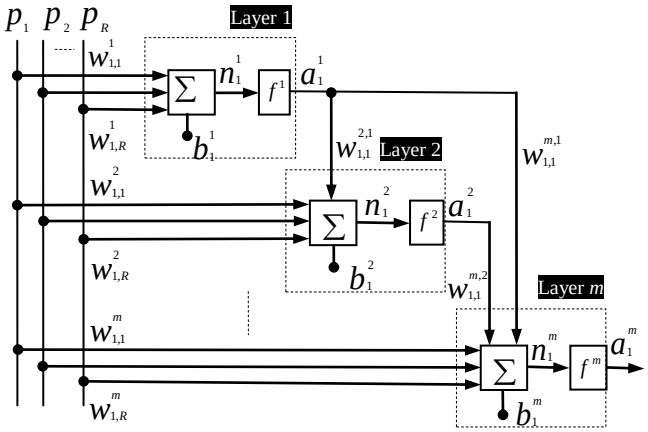


Fig.1. SNC-NN with multiple input and single output

$$P_C = \underbrace{\sum_{m=1}^{M}\sum_{q=0}^{m-1} S^m S^q}_{\text{Weights}} + \underbrace{\sum_{m=1}^{M} S^m}_{\text{Biases}} \quad (1)$$

where,

P -Input vector, $p = [1, 2, \ldots R]$

S^m -Number of neurons in the layer, $m = [1,2,\ldots M]$ and $S^o = p$

$w^m_{i,R}$ -Input weight of neuron 'i' of layer 'm' for external input 'R'

$w^{m,k}_{i,j}$ -Link weight of neuron 'i' of layer 'm' for input from neuron 'j' of layer 'k'

b^m_i -Bias for neuron 'i' of layer 'm'

f^m -Activation functions of all neurons in a layer 'm'

a_i^m -Output of neuron '*i*' of layer '*m*'

The structure of the SNC-NN architecture is denoted as, $S^0 - \sum_{i=1}^{M-1} S^i (h) - S^M$, where *h* is the number of hidden layers with single neuron in each layer.

2.1.2 Feed-forward Architecture:

Feed-forward architecture for multi-input and single-output system is shown in Fig.2. It has an input layer, one or more hidden layers and output layer. Each neuron model in the architecture includes a nonlinear activation function and those in output layer use pure-linear function. Unlike SNC, in this network, the input signal is restricted only to the input layer and propagates through layers of network in a forward direction. Feed-forward architecture with one hidden layer is called as Single Layer Feed Forward Neural Network (SLFF-NN) and when multiple layers are used it is called Multilayer Layer Feed Forward Neural Network (MLFF-NN).

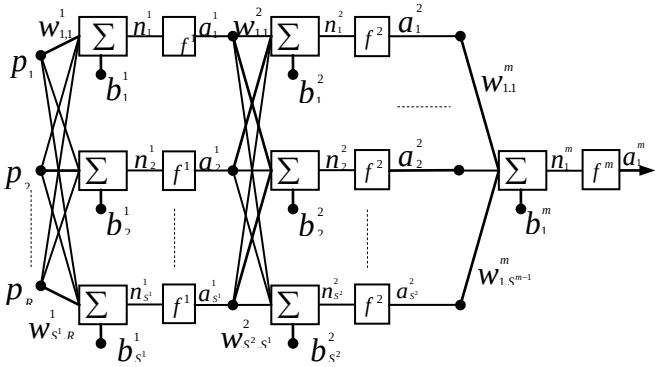


Fig.2. Feed-forward Network with multiple inputs and single output

The total number of parameters (P_{MLFF}) in the multilayer feed-forward architecture can be obtained from Eq.(2). As SLFF-NN is a special case of MLFF-NN with one hidden layer, the same formula suits both type of FF-NN. The first and second terms in Eq.(2) are deduced considering separately the weights and biases respectively.

$$P_{MLFF} = \underbrace{\sum_{m=1}^{M} S^{m-1} S^m}_{\text{weights}} + \underbrace{\sum_{m=1}^{M} S^m}_{\text{biases}} \quad (2)$$

where, w_{ij}^m - Interconnection weight of neuron '*i*' of layer 'm' for input from neuron '*j*' of layer '(*m*-1)'.The structure of the FF-NN architecture is denoted as $S^0 - S^1 - ..... - S^M$.

2.2 STRUCTURAL COMPACTNESS AND COMPUTATIONAL COMPLEXITY OF NN MODEL

The structure of neural network model depends on the number of inputs, number of outputs and the degree of nonlinearity of the system. The number of neurons in the input/output layer is uniquely defined and is equal to that of inputs/outputs of the system to be modeled. The number of hidden layers, hidden neurons and the type of architecture are the choice of the design for a desired accuracy.

The number of parameters and nonlinear function extractions required by the network indicates its computational requirements. The number of neurons per layer is an index of complexity as it requires simultaneous computation and directly linked to the hardware complexity. The number of parameters per neuron is an index of its mapping capability. Each parameter warrants some mathematical operations. The relationship between the number of hidden neurons and the number of parameters of the network demonstrates the relationship between structural compactness and mathematical complexity for a given architecture. This relationship called as parameter growth curve obtained from parametric analysis using Eq.(1) and Eq.(2) is shown in Fig.3.

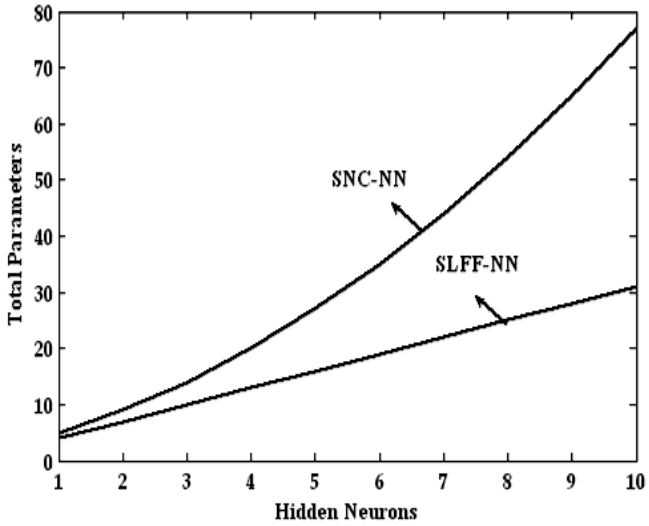


Fig.3. Parameter Growth Curve

Parametric growth curve is a new concept and applied to find the novel reasons for the network model to be compact and simple to implement. From the parametric growth curve of the neural network models, it is interesting to observe that for a SNC-NN the parameter growth is exponential, whereas it is linear for SLFF-NN/ MLFF-NN. The parameters in SNC-NN increase more rapidly with the increase in the total number of hidden neurons as compared to SLFF-NN and MLFF-NN. SNC-NN attains a massive interconnected multilayer structure with higher degree of non-linearity. This feature would allow the SNC-NN to reach the desired solution with lesser number of hidden neurons resulting in more compact models. This study and parametric analysis indicates that for a given nonlinear function the SNC-NN would result in more compact model than SLFF/MLFF-NN. This theoretical fact is further substantiated by the results of benchmark examples presented in section 3.

The structure of neural network model depends on the number of inputs, number of outputs and the degree of nonlinearity of the system. The number of neurons in the input/output layer is uniquely defined and is equal to that of inputs/outputs of the system to be modeled. The number of hidden layers, hidden neurons and the type of architecture are the choice of the design for a desired accuracy.

2.3 LEARNING ALGORITHMS

There are different types of learning algorithms available in the literature to train the NN [12]. Among the directed search algorithms, the steep descent algorithm is a first order approach. Many variants of steepest descent algorithms are popularly used for training the NN. The first order approach provides simplicity, but is not suitable when high degree of accuracy is required. A significant improvement in accuracy is obtained when the training algorithm is based on second order approach namely Newton's method, Gauss Newton's method, and Levenberg-Marquardt (LM) method [12]. The higher accuracy is obtained at the cost of increased complexity of update laws. The LM algorithm is widely accepted as the most efficient one in achieving good accuracy [12].

In this paper, a number of Benchmark examples are chosen to compare the structural compactness and computational complexity of the SNC-NN models for function approximation. For comparison, all such models are to be trained using the same learning algorithm. As LM algorithm is best suited for offline training it is used to train all the neural network models.

3. COMPARISON OF NN MODELS FOR BENCHMARK NON-LINEAR FUNCTIONS

The distinct advantages claimed for the SNC-NN in this paper are made evident using nonlinear benchmark examples with different types and degrees of nonlinearity [3]. The nonlinear plants as benchmark examples (I to IV) are modeled using neural networks. The SNC-NN, SLFF-NN and MLFF-NN architectures are considered for performance comparison. The uniformity is maintained among the models by adopting the same type of activation functions for the neurons of the various layers, learning algorithm, training/test data and training target MSE. All NN models use tan-sigmoid/ linear activation function for hidden layers/output layer and LM learning algorithm without weight freezing. The simulations are carried out using MATLAB 7. The SNC-NN and SLFF-NN are self organizing, as one neuron can be added at a time and training carried out to meet the target MSE. This systematic procedure helps in automation of the training process, and greatly reduces the design time.

For the MLFF-NN the determination of the number of layers and number of neurons in each layer is heuristic and hence the design is not systematic and the architecture is not unique. Therefore the MLFF-NN modeling does not render itself to automation in design. However, in this paper for the purpose of comparison the architecture proposed by Narendra et al [3] is chosen for the four benchmark examples and re-trained using LM algorithm. The MSE achieved for the four Benchmark examples are chosen as target MSE for SNC-NN and SLFF-NN model.

For control and estimation problems the complexity of the model assumes importance as the computation/ estimation time has to be small enough for effective control. The mathematical complexity is compared by determining the number of basic operations needed by the model. This will depend upon the architecture. Let N_a be the number of additions, N_m be the number of multiplications and N_{nf} be the number of nonlinear function extractions needed for the model. The time taken in real time by a processor for a given model can be easily computed if the time for the basic operations is known. Let t_a, t_m and t_{nf} be the execution time needed for addition, multiplication and nonlinear function extraction. The total execution time T_{total} can be obtained as,

$$T_{total} = N_a \times t_a + N_m \times t_m + N_{nf} \times t_n \quad (3)$$

This general approach helps to determine the execution time for any chosen target digital hardware such as ADSP /FPGA. The applications for this study would demand one or more digital signal processors. Hence in this paper ADSP-TS101 with operating clock frequency of 250 MHz is used for implementing the NN models. The execution time in micro seconds for the operations namely addition, multiplication, and non-linear function extraction are presented in Table.1 [16].

Table.1. Required Time to Execute Mathematical Functions

Mathematical operation	Required Execution
Addition	0.004
Multiplication	0.004
Tan-Sigmoid $\{(e^n - e^{-n})/(e^n + e^{-n})\}$	0.224

3.1 RESULTS AND DISCUSSION FOR BENCHMARK EXAMPLES

The performance of the NN models for the four bench mark examples is presented. For each example three NN models namely SNC-NN, SLFF-NN and MLFF-NN are built using the previously described design procedure using the same training data and target MSE. To aid comparison of all the NN models, the testing MSE, number of neurons, execution time using ADSP-TS101 is obtained and presented.

Example-1: The plant is described as in Eq.(4) and Eq.(5). The Neural Network model has single input $u(k)$ and a single output $f(u)$.

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + f[u(k)] \quad (4)$$

$$f(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u) \quad (5)$$

The function $f(u)$ is a Single Input and Single Output (SISO) model. Training data of 500 points are generated assuming a random input signal "$u(k)$" uniformly distributed in the interval [-1 1]. All the three NN models are obtained for an accuracy of 1×10^{-15}. All the models are tested for a test input signal $u(k) = \sin(2\pi k/250)$ for $k \leq 167$ and $u(k) = 0.5\sin(2\pi k/250) + 0.5\sin(2\pi k/25)$ for $k > 167$. The testing MSE obtained, architecture, number of neurons and computations required for the three models are summarized in Table.2. The performance of SNC-NN model for the testing data is plotted and shown in Fig.4(a) and the error is shown in Fig.4(b).

Example-2: The plant is described as in Eq.(6) and Eq.(7). The NN model has two inputs $y(k)$, $y(k-1)$ and a single output $f[y(k),y(k-1)]$.

$$y(k+1) = f[y(k),y(k-1)] + u(k) \quad (6)$$

$$f\left[y(k),y(k-1)\right] = \frac{y(k)y(k-1)\left[y(k)+2.5\right]}{1+y^2(k)+y^2(k-1)} \quad (7)$$

Training data of 500 points are generated from the plant

model assuming a random input signal "$u(k)$" uniformly distributed in the interval [-2 2]. All the models are trained for a target MSE of 1×10^{-13}.The architecture, testing MSE, number of neurons and computations required for all models is shown in Table.2.

The performance comparison of actual plant model and SNC-NN based plant model for a test input signal $u(k)$ = $\sin(2\pi k/25)$ is shown in Fig.5(a) and the error in Fig.5(b).

Example-3: The plant is described as in Eq.(8) to Eq.(10). Two NN models are used to model unknown nonlinear functions $f[y(k)]$ and $g[u(k)]$ and the inputs are $y(k)$, and $u(k)$ respectively.

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \quad (8)$$

$$f\left[y(k)\right] = \frac{y(k)}{1+y^2(k)} \quad (9)$$

$$g[u(k)] = u^3(k) \quad (10)$$

Training data of 500 points are generated from the plant model assuming a random input signal "$u(k)$" uniformly distributed in the interval [-2 2]. Two separate SNC-NN, SLFF-NN and MLFF-NN models are trained for the training MSE of 1×10^{-15}. The architecture, testing MSE, number of neurons and computations required for models is shown in Table.2.

The performance comparison of actual plant model and SNC-NN based plant model for a test input signal $u(k)$ = $\sin(2\pi k/25)$ + $\sin(2\pi k/10)$ is shown in Fig.6(a) and error in Fig.6(b).

Example-4: The plant is described in Eq.(11) and Eq.(12). The NN model has five inputs $y(k)$, $y(k\text{-}1)$, $y(k\text{-}2)$, $u(k)$, $u(k\text{-}1)$ and single output $f[x_1,x_2,x_3,x_4,x_5]$.

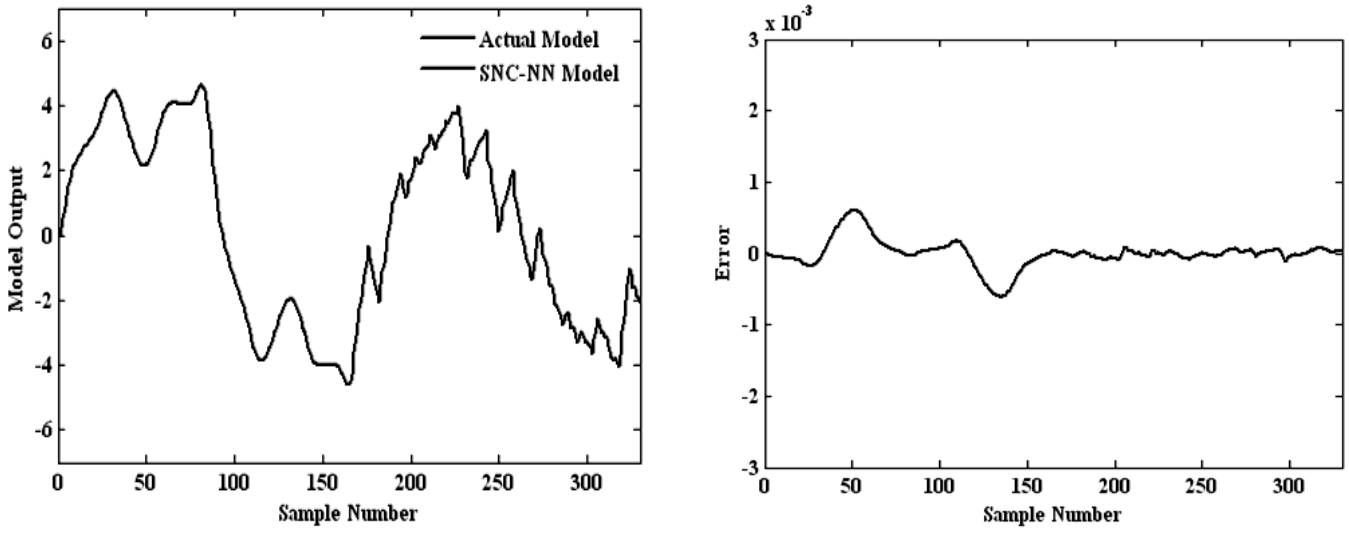
$$y(k+1) = f[y(k),y(k-1),y(k-2),u(k),u(k-1)] \quad (11)$$

$$f\left[x_1,x_2,x_3,x_4,x_5\right] = \frac{x_1x_2x_3x_4x_5(x_3-1)+x_4}{1+x_3^2+x_2^2} \quad (12)$$

Training data of 500 points are generated from the plant model assuming a random input signal "$u(k)$" uniformly distributed in the interval [-1 1]. The three models are trained for a target MSE of $1\times10^{-8.}$ The architecture, testing MSE, number of neurons and computations required for models is shown in Table.2.

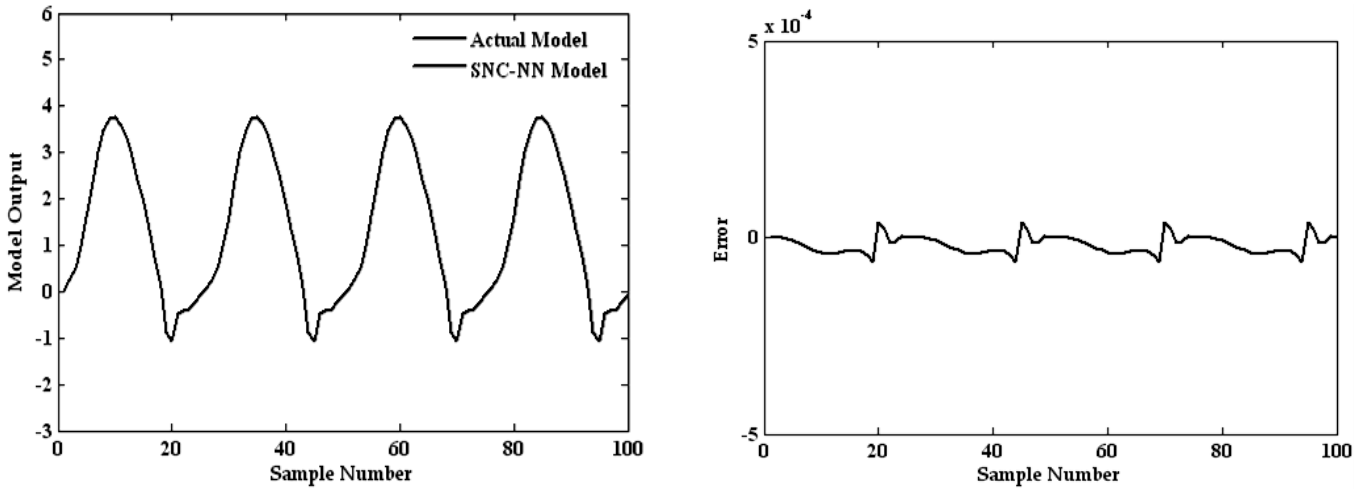
The performance of actual plant model and SNC-NN based plant model for a test input signal $u(k)$ = $sin(2\pi k/250)$ for $k \leq 500$ and $u(k)$ = $0.8\sin(2\pi k/250) + 0.2\sin(2\pi k/25)$ for $k > 500$ is shown in Fig.7(a) and error in Fig.7(b).

The performance, complexity and computations for all the NN models (Examples I-IV) are presented in Table.2. The results show that the testing MSE is nearly same and proves that each NN model is built to maintain the desired accuracy for the respective examples. As the nonlinear function is the most complex part of the computation it can be observed that the computational complexity is proportional to the total number of hidden neurons. NN models implemented in DSP processor executes sequentially. Hence total execution time is calculated using Eq.(3) and taken as reference to compare the performance of chosen NN Model.



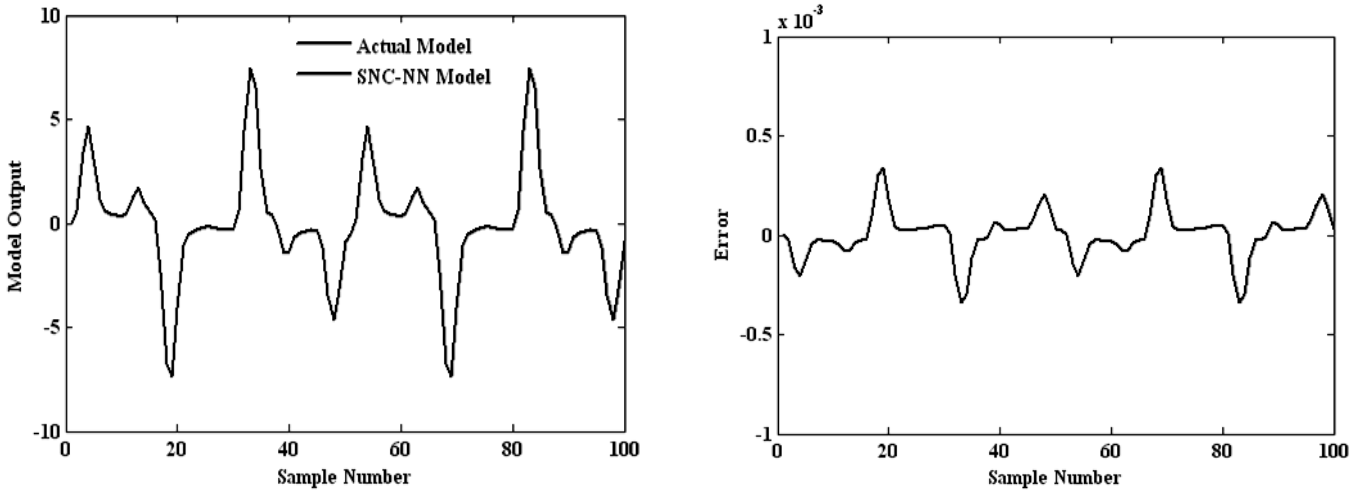
(a) (b)

Fig.4. Example-I(a) SNC-NN Model output compared with actual output (b) Error Curve



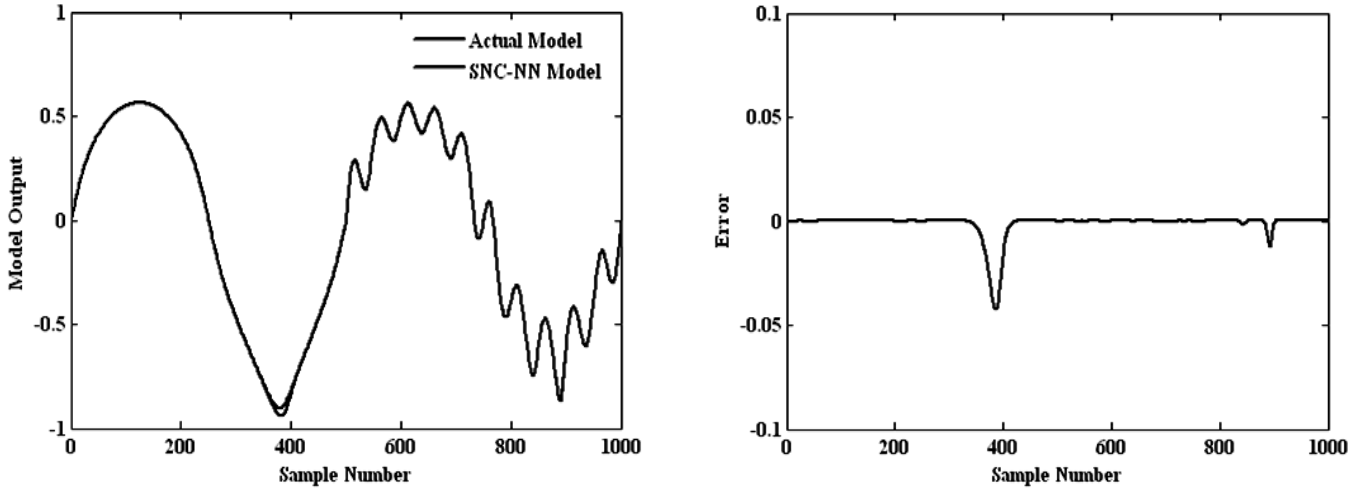
(a) (b)

Fig.5. Example-II (a) SNC-NN Model output compared with actual output (b) Error Curve



(a) (b)

Fig.6. Example-III (a) SNC-NN Model output compared with actual output (b) Error Curve



(a) (b)

Fig.7. Example-IV(a) SNC-NN Model output compared with actual output (b) Error Curve

Table.2. Comparison of NN Models for Benchmark Examples

Benchmark Examples	NN Model Performance				NN Parameters and Computational complexity					
	Models	Architectures		Testing MSE of Plant Output	No. of hidden Neurons	No. of Parameters	No. of Additions	No. of Multiplication	No. of Tan sigmoid	Execution Time (μs)
I	**SNC-NN**	1-13(h)-1		3.90×10^{-8}	**13**	**119**	105	105	**13**	**03.75**
	SLFF-NN	1-26-1		3.90×10^{-8}	26	79	52	52	26	06.24
	MLFF-NN	1-20-10-1		3.87×10^{-8}	30	261	230	230	30	08.56
II	**SNC-NN**	2-15(h)-1		$\mathbf{8.71\times10^{-10}}$	**15**	**168**	152	152	**15**	**04.57**
	SLFF-NN	2-132-1		1.54×10^{-2}	132	529	396	396	132	32.73
	MLFF-NN	2-20-10-1		1.21×10^{-7}	30	281	250	250	30	08.72
III	**SNC-NN**	$f[y(k)]$	1-5(h)-1	1.46×10^{-8}	**9**	**47**	36	36	**9**	**02.39**
		$g[u(k)]$	1-4(h)-1							
	SLFF-NN	$f[y(k)]$	1-15-1	1.45×10^{-8}	30	92	60	60	30	07.46
		$g[u(k)]$	1-15-1							
	MLFF-NN	$f[y(k)]$	1-20-10-1	1.45×10^{-8}	60	522	460	460	60	17.62
		$g[u(k)]$	1-20-10-1							
IV	**SNC-NN**	5-14(h)-1		$\mathbf{3.79\times10^{-5}}$	**14**	**195**	180	180	**14**	**04.57**
	SLFF-NN	5-49-1		6.79×10^{-4}	49	344	294	294	49	13.32
	MLFF-NN	5-20-10-1		3.67×10^{-4}	30	341	310	310	30	09.20

The unique structural growth of SNC architecture enables automation in design similar to a single layer network. The cascading of neurons increases the order of neurons and layers and this aspect increases the mapping capability. The complex nonlinear functions are modeled accurately similar to a multilayer network and mostly even better. Hence it is proved and shown through the parametric analysis that the SNC-NN combines the advantages of both single and multilayer networks resulting in a compact architecture that can self-train and self-grow to meet the desired target.

For all the nonlinear Benchmark examples, the number of neurons required is minimum (14, 16, 11, 15) for SNC-NN model as against maximum (31, 31, 62, 31) for FF-NN based models. The execution time required is always minimum for SNC-NN model. Hence the SNC-NN is superior for modeling highly non linear functions and delivers a model which is most compact and executes faster. These features are preferred for real time applications. One such application is investigated and presented in the following section.

4. PRACTICAL REAL TIME EXAMPLE

To further illustrate the effectiveness of SNC-NN, the practical real time example taken in this paper is the NN based modeling of waveform processing and filtering technique used in control and estimation of power electronic converters and AC drives [17].

The power electronic converters and variable speed AC drive systems often deal with complex voltage and current waves that are rich in harmonics. It is often necessary to retrieve the fundamental component of these waves in order to calculate, for example, the displacement power factor (DPF), fundamental frequency active and reactive power, and energy measured by a kilowatt-hour meter. The NN based fundamental component estimator has many advantages as listed in [17]. In this paper, the fundamental component estimator is modeled using NN for a simple single-phase square wave inverter, which finds application mainly in uninterruptible power supply (UPS) system.

Training data of around 5000 points are generated from the single-phase inverter circuit. The input to the NN cannot be chosen as square wave output of the inverter alone because the square wave amplitude is constant in the half-cycle, the NN cannot generate a continuously variable sine wave directly from the square wave [17]. For this reason, an auxiliary input wave is generated from the square wave through a first order low pass filter. The block diagram of NN based waveform processor is shown in Fig.12. The inputs to NN are inverter output v_a (square wave) and filter output v'_a. The output of NN is the desired fundamental component of the inverter output voltage v_{af}.

In this paper for the purpose of comparison the architecture (2-15-15-1) proposed by Jin Zhao et al [17] is chosen and re-trained using LM algorithm. The MSE achieved for the 2-15-15-1 is chosen as target MSE for SNC-NN and SLFF-NN model.

The target MSE chosen is 1×10^{-8} for training all the NN models. The tansigmoid functions are chosen for hidden layers

and pure linear function is chosen for output layer. The SNC-NN, SLFF-NN and MLFF-NN are trained with LM algorithm without weight freezing for the same accuracy from the same input/output data. For the SNC-NN model, it is observed that the target MSE is reached with 10 hidden layers. For SLFF-NN, it is found that even after adding 400 hidden neurons, the target MSE reached is 2.28×10^{-7}. Therefore, SLFF-NN still requires more number of neurons to reach the desired target MSE.

All the three offline trained networks are tested for online estimation of fundamental component from the harmonic rich output waveform of the inverter. The inputs of the SNC-NN based waveform estimator are shown in Fig.9. The actual output, estimated value and error are shown in Fig.10.

From the Table.3, it is found that the testing MSE for all the three models are nearly same. The SNC-NN model is most compact model with only 10 hidden neurons as compared to SLFF-NN and MLFF-NN model that requires 400 and 30 hidden neurons respectively. The computation time required for all the models on implementation is presented in Table.3. It can be observed that the SNC-NN model is found to be 34.73 times faster than SLFF-NN model and 3.10 times faster than MLFF-NN model. The SNC-NN based model is the most compact in structure and less complex in computation that enables easy implementation and faster execution.

For the practical example, implementation in reconfigurable digital hardware is also investigated. FPGA allows parallel implementation of neurons in a layer and hence best suited for NN applications. From the literature [18],[19] the best possible implementations of Single precision floating point arithmetic operations on Virtex-II Device is reported. The slice requirements and cycles achievable are shown in Table.4.

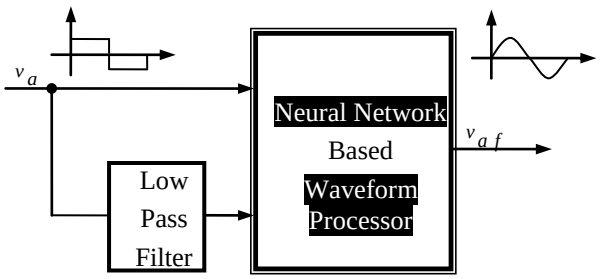


Fig.8. Inputs of NN based Waveform Processor

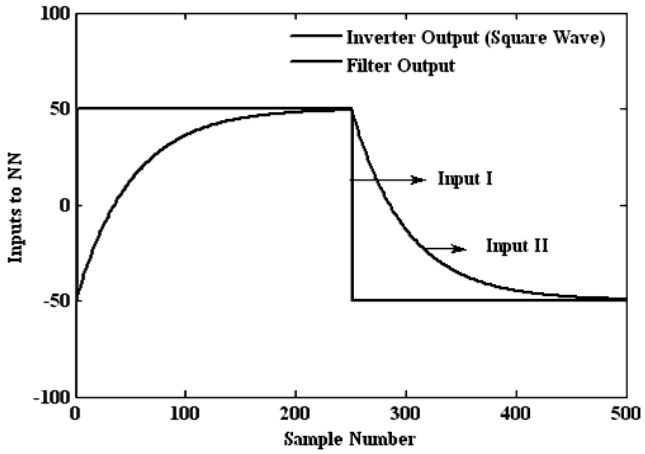
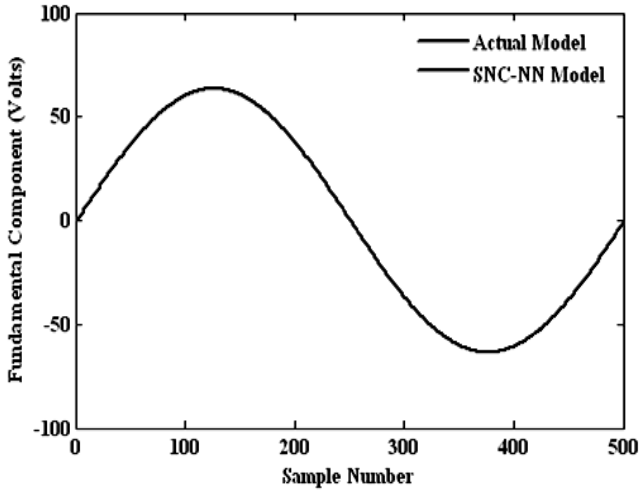
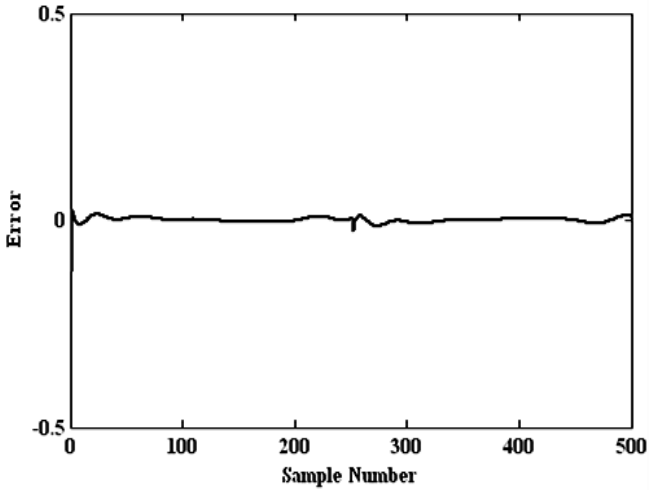


Fig.9. Inputs to the SNC-NN model for waveform estimator



(a)



(b)

Fig.10. Performance of the SNC-NN Model for Waveform Estimator (a) The actual and estimated fundamental waveforms (b) Error between the actual and estimated fundamental waveforms

The issues in FPGA implementation of NN is the contradicting requirements of resource and execution time. Faster execution time requires more parallelism and hence more resource leading to higher cost. Hence a compromise to obtain a low cost high speed implementation is to minimize the product of resource and execution time ($P_{r\times t}$). This can be used as a figure of merit to compare the performance of various implementations.

Table.3. Performance and Parameters of NN Waveform Estimator for ADSP Implementation

Real Time Practical Examples	NN Model Performance			NN Parameters and Computations					
	NN Models	NN Architectures	Testing MSE	No. of Hidden Neurons	No. of parameters	No. of Addition	No. of Multiplication	No. of Tansigmoid	Execution Time (μs)
Waveform Processing	**SNC-NN**	**2-10(h)-1**	**6.21×10^{-5}**	**10**	**88**	**77**	**77**	**10**	**2.856**
	SLFF-NN	2-400-1	1.56×10^{-4}	400	1601	1200	1200	400	99.20
	MLFF-NN	2-15-15-1	4.82×10^{-5}	30	301	270	270	30	8.88

Table.4. Single Precision Floating Point operations on Virtex-II 6000-5 Device

Arithmetic Operations	Slices	Cycles
Addition	496	1
Multiplication	598	1
Division	1929	1
Exponent	5564	74
Tansigmoid	8485	78

To minimize the product of resource and execution time the concept of layer multiplexing is adopted [20]. The parallel operation of neurons in a layer is preserved to reduce execution time, but the sequential operation of layers is exploited to reduce resource. Using the above approach a single neuron with maximum number of inputs is chosen for SNC NN. For MLFF-NN the single largest layer is implemented. As the tan sigmoid function requires large resource/slices, only one computing block is implemented for the NN model and used sequentially in a layer. This method yields minimum (P_{rxt}) for each of the NN models. The execution cycles and slices for the three models are shown in Table.4 for comparison.

From the Table.5 it is evident that the SNC NN architecture requires the minimum $P_{r\times t}$. Hence it derives the most compact and a fast NN model that is distinct compared to SLFF-NN and MLFF-NN on FPGA implementation.

The performance evaluation using real time waveform processing and its implementation on ADSP and FPGA also confirms the superiority of SNC-NN based models for such real time applications.

Table.5. Performance of NN Waveform Estimator for FPGA Implementation

Real Time Practical Examples	NN Models		NN Implementation on FPGA		
	NN Models	NN Architecture	Slices (S)	Cycles (C)	Resource time Product (P_{rxt})
Waveform Processing	**SNC-NN**	**2-10(h)-1**	**41.10**	**802**	**32967.01**
	SLFF-NN	2-400-1	1777.106	31204	55452815.62
	MLFF-NN	2-15-15-1	512.33	2346	1201928.52

5. CONCLUSION

A compact as well as accurate model assumes importance in real time applications such as control and estimation tasks. The model has to be easy to design, simple to implement and fast to respond with good accuracy. This paper identifies and proposes a single neuron cascaded neural network (SNC-NN) to meet these desired requirements. Using four benchmark nonlinear examples and one practical real time example the SNC-NN, SLFF-NN and MLFF-NN models are built for the same target MSE. The performance of all the three NN models presented in Tables proves that the SNC-NN requires lesser number of neurons and is superior in terms of compactness and complexity as compared to FF-NN models.

The computational requirements and execution time on implementation using ADSP and FPGA is presented. From the results of Table.2 SNC-NN model is shown to be compact and distinctly faster for all the benchmark examples. The performance of the SNC-NN model for the practical/real time

application of waveform processing is found to be efficient for FPGA implementation.

The SNC-NN model offers a promising solution for estimators and controllers that demands higher accuracy and fast execution. The various results highlight the capability and suitability of SNC-NN model for real time applications which is the major conclusion and contribution of this paper. In future such SNC-NN would be realizable as ASIC chips and used for a wide variety of engineering applications.

ACKNOWLEDGMENT

The authors acknowledge the financial support of the Department of Science and Technology, Delhi for the grant of Junior Research Fellowship (JRF)-Professional to the first author for pursuing this research work. The research project titled "AI techniques for Electrical Drives" is supported by the grants from the All India Council for Technical Education (AICTE), a statutory body of Government of India. File Number: No 8023/BOR/RID/RPS-79/2007-08 and 8020/RID/TAPTEC-32/2001-02.

REFERENCES

[1] K. Hornik, M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", *IEEE Transactions on Neural Networks*, Vol. 2, pp. 359-366, 1989.

[2] K. Hornik, M. Stinchcombe and H. White, "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks", *IEEE Transactions on Neural Networks*, Vol. 3, pp. 551-560, 1990.

[3] K.S. Narendra, and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 4-27, 1990.

[4] Chien-Cheng Lee, Pau-Choo Chung, Jea-Rong Tsai, and Chein-I Chang, "Robust Radial Basis Function Neural Networks", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, Vol. 29, No. 6, pp. 674-685, 1999.

[5] S.E. Fahlman, and C. Lebiere, "The Cascade-Correlation Learning Architecture", *CMU Computer Science Dept Technical Report*, CS-90-100, pp. 1-13, 1991.

[6] Robert J. Schilling, James J. Carroll, and Ahmad F. Al-Ajlouni, "Approximation of Nonlinear Systems with Radial Basis Function Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 12, No. 1, pp. 1-15, 2001.

[7] J.C. Patra, and A.C. Kot, "Nonlinear Dynamic System Identification Using Chebyshev Functional Link Artificial Neural Networks", *IEEE Transactions on Systems, Man, Cybernetics Part B: Cybernetics*, Vol. 32, No. 4, pp. 505-511, 2002.

[8] Tsu-Tian Lee and Jin-Tsong Jeng, "The Chebyshev-Polynomials-Based Unified Model Neural Networks for Function Approximation" *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, Vol. 28, No. 6, pp. 925-935, 1998.

[9] Aydogan Savran, "Multifeedback-Layer Neural Network", *IEEE Transactions on Neural Networks*, Vol. 18, No. 2, pp. 373-384, 2007.

[10] Chengyu Gan, and Kourosh Danai, "Model-Based Recurrent Neural Network for Modeling Nonlinear Dynamic Systems", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, Vol. 30, No. 2, pp. 344-351, 2000.

[11] Liying Ma, and K. Khorasani, "Constructive Feedforward Neural Networks Using Hermite Polynomial Activation Functions", *IEEE Transactions on Neural Networks*, Vol. 16, No. 4, pp. 821-833, 2005.

[12] Martin T.Hagan, Howard B. Demuth, and Mark Beale, "*Neural Network Design*", Cengage Learning (Thompson), 2008.

[13] G. A. Rovithakis, I. Chalkiadakis, and M. E. Zervakis, "High-Order Neural Network Structure Selection for Function Approximation Applications Using Genetic Algorithms", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, Vol. 34, No. 1, pp. 150-158, 2004.

[14] Sheng Chen, Xia Hong, Bing L. Luk, and Chris J. Harris, "Construction of Tunable Radial Basis Function Networks Using Orthogonal Forward Selection", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, Vol. 39, No. 2, pp. 457-466, 2009.

[15] Nicholas K.Treadgold, and Tamás D. Gedeon, "Exploring Constructive Cascade Networks", *IEEE Transactions on Neural Networks*, Vol. 10, No. 6, pp. 1335-1350, 1999.

[16] Steven Smith, "*Digital Signal Processing: A Practical Guide for Engineers and Scientists*", Newnes, 2002.

[17] JinZhao, and Bimal K. Bose, "Neural-Network-Based Waveform Processing and Delayless Filtering in Power Electronics and AC Drives", *IEEE Transactions on Industrial Electronics*, Vol. 51, No. 5, pp. 981-991, 2004.

[18] Keith Underwood Sandia, "FPGA vs. CPUs: Trends in Peak Floating-Point Performance", *Proceedings of 12th International Symposium on Field Programmable Gate Arrays*, pp. 171-180, 2004.

[19] Jérémie Detrey, Florent de Dinechin, "A parameterized floating-point exponential function for FPGAs", *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 27-34, 2005.

[20] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feed-forward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization", *IEEE Transactions on Neural Networks*, Vol. 18, No. 3, pp. 880-888, 2007.