Int. J. Appl. Math. Comput. Sci., 2008, Vol. 18, No. 3, 369–375
DOI: 10.2478/v10006-008-0033-8

DETERMINING THE WEIGHTS OF A FOURIER SERIES NEURAL NETWORK ON THE BASIS OF THE MULTIDIMENSIONAL DISCRETE FOURIER TRANSFORM

KRZYSZTOF HALAWA

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50–370 Wroclaw, Poland
e-mail: krzysztof.halawa@pwr.wroc.pl

This paper presents a method for training a Fourier series neural network on the basis of the multidimensional discrete Fourier transform. The proposed method is characterized by low computational complexity. The article shows how the method can be used for modelling dynamic systems.

Keywords: orthogonal neural networks, Fourier series, fast Fourier transform, approximation, nonlinear systems.

1. Introduction

Fourier series neural networks (FSNNs) belong to the class of orthogonal neural networks (ONNs) (Zhu *et al*., 2002; Sher *et al*., 2001; Tseng and Chen, 2004; Rafajłowicz and Pawlak, 1997). They are feedforward networks, similarly to sigmoidal neural networks.

The inputs of an MISO (multiple-input single-output) FSNN are connected with neurons having orthogonal harmonic activation functions from the trigonometric series $c_0,\ c\cdot\sin(\alpha u),\ c\cdot\cos(\alpha u),\ \alpha=1,2,3,\ldots,$ where c_0 and c are constants. These neurons have solely a single input.

Multiplying nodes calculate products for all combinations of output signals from the aforementioned neurons. The linear neuron sums the weighted outputs of the multiplying nodes yielding the output of the network. The weights of this neuron are subjected to changes during the network learning process. The network architecture is designed so that the same transformation as in a multidimensional Fourier series could be achieved. An MIMO (multiple-input multiple-output) FSNN may be created by joining additional linear neurons to the output layer. The number of linear neurons equals the number of network outputs. In order to get a simple notation, in this paper (except Section 4), a case is considered wherein each input of the neural network is interconnected with the same number of orthogonal neurons. If each input is associated with a different number of neurons, the procedure is analogous. The procedure to be used in such a situation is also described here. The orthogonal neural network with a single output is shown in Fig. 1. In this figure, O denotes orthogonal neurons, Π represents product nodes, and Σ is the linear neuron. If each input of a network is associated with $N-1$ harmonic neurons, where N is an even number and $c=1$, $c_0=1$, the network output is given by

$$y=\left[w_1,w_2,\ldots,w_{(N-1)^S}\right]\cdot\left(\begin{bmatrix}1\\ \sin(u_1)\\ \cos(u_1)\\ \sin(2u_1)\\ \cos(2u_1)\\ \vdots\\ \sin((N/2-1)u_1)\\ \cos((N/2-1)u_1)\end{bmatrix}\otimes\begin{bmatrix}1\\ \sin(u_2)\\ \cos(u_2)\\ \sin(2u_2)\\ \cos(2u_2)\\ \vdots\\ \sin((N/2-1)u_2)\\ \cos((N/2-1)u_2)\end{bmatrix}\right.$$
$$\left.\otimes\cdots\otimes\begin{bmatrix}1\\ \sin(u_S)\\ \cos(u_S)\\ \sin(2u_S)\\ \cos(2u_S)\\ \vdots\\ \sin((N/2-1)u_S)\\ \cos((N/2-1)u_S)\end{bmatrix}\right), \tag{1}$$

where $\otimes$ denotes the Kronecker product, S is the

number of network inputs, $w_1, w_2, \ldots, w_{(N-1)^S}$ are network weights, $u_1, u_2, \ldots, u_S$ are network inputs, $1, \sin(ku_i), \cos(ku_i)$ are the activation functions of the neurons connected to the i-th network input $k = \{1, 2, \ldots, N/2\}$, y is the network output. What is worth mentioning is that the values we get upon calculating Kronecker's products inside the round brackets in (1) are equal to the outputs of multiplying nodes. The input values to the network must be within the interval $[0, 2\pi)$.

FSNNs have numerous advantages which are unavailable in sigmoidal networks, e.g.,

- high-speed convergence of the learning process, if the cost function is the mean squared error, through the lack of local minima for the gradient descent algorithm,
- the output is linear with respect to the weights,
- the relationship between the number of inputs and outputs and the maximum number of orthogonal neurons is well known.

The comparison of the learning speeds between the two-input-one-output FSNN having 30 harmonic neurons, which was trained with a gradient descent algorithm, and a 2-10-1 sigmoidal network, which was trained with the Levenberg-Marquardt method, is depicted in Fig. 2. The networks were trained to map the function $f(x_1, x_2) = \exp\left(-(x_1-2)^2 - (x_2-1.5)^2\right) - \exp\left(-(x_1-4)^2 - (x_2-4)^2\right)$. The training set was generated according to the equation $y_m = f(x_{1m}, x_{2m}) + 0.5 \cdot e_m$, where e_m are normally distributed random numbers with zero mean and unit standard deviation, x_{1m} and x_{2m} are random numbers uniformly distributed on the interval $[0, 2\pi)$, $m = \{1, 2, \ldots, 1000\}$. Learning results were evaluated on a testing set composed of 2000 elements. It had been generated in an analogous way as that for the training set. In the case of an FSNN, the values of the weights and output signals from neurons may easily take physical interpretations, which is of importance when these networks are used in data stream processing.

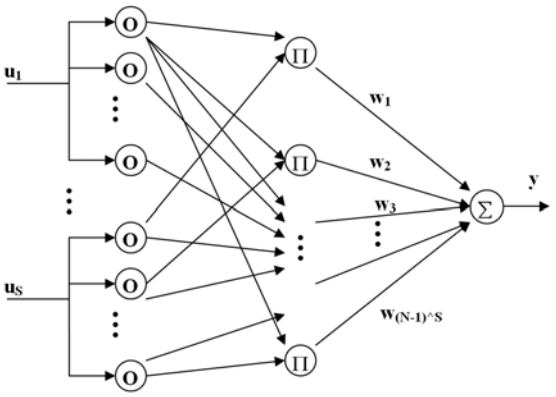

Fig. 1. MISO structure of an orthogonal neural network.

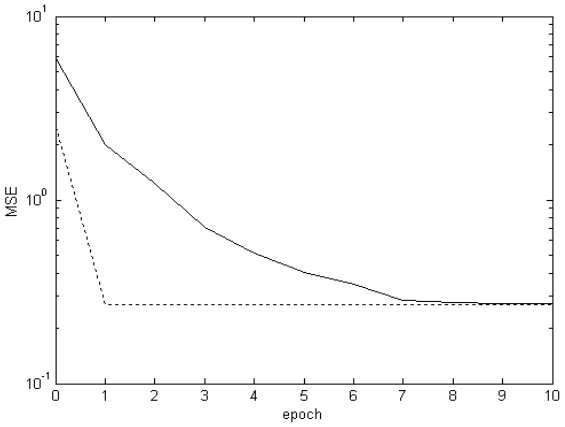

Fig. 2. Comparison of learning process speeds between the FSNN having 30 orthogonal neurons which was trained with a gradient descent algorithm (dotted line) and a sigmoidal 2-10-1 network which was trained with a Levenberg-Marquardt algorithm (solid line). MSE denotes the mean squared error.

The paper shows how to calculate the weights $w_1, w_2, \ldots, w_{(N-1)^S}$ on the basis of the S-dimensional discrete Fourier transform (DFT) (Gonzalez and Woods, 1999), which is obtained by the fast Fourier transform (FFT) (Bracewell, 1999; Chu and George, 2000; Walker, 1996; Van Loan, 1992). The method set forth here enables us to calculate weight values quickly and efficiently. A similar method for a one-dimensional case was described in (Rafajłowicz and Skubalska-Rafajłowicz, 1993). The remainder of this work is organized as follows: Section 2 provides the method used to determine the weights on the basis of an S-dimensional DFT. Two ways of applying the proposed algorithm when input data are not evenly spaced are also shown there. The network pruning is described. Section 3 is devoted to the computational complexity of the proposed method. Section 4 contains necessary modifications of the presented method which have to be carried out when each input is associated with a different number of neurons.

2. Calculating the weights on the basis of the S-dimensional DFT

In what follows we describe how the weights are determined on the basis of the S-dimensional DFT when the network should approximate a function $f(X)$, $X \in \mathbb{R}^S$ which is piecewise continuous and satisfies Dirichlet conditions.

The presented method can be applied directly if the training set of values $\{y_m, U_m \cdot p\}$, $m = \left\{1, 2, \ldots, N^S\right\}$,

$y_m \in \mathbb{R}$, $U_m \in \mathbb{R}^S$ is at our disposal, where $y_m = f_e(U_m \cdot p) = f(U_m \cdot p) + \varepsilon_m$, $p = 2\pi/N$, ε_m are uncorrelated random variables with zero expectations and finite variances, the components of U_m multiplied by p are evenly spaced in a hypercube and are equal to $\left\{0, 1\frac{2\pi}{N}, 2\frac{2\pi}{N}, \ldots, (N-1)\frac{2\pi}{N}\right\}$.

Necessary modifications that have to be made if $U_m \cdot p$ are not evenly spaced in a hypercube with side length 2π are described at the end of this section. This situation is typical for a passive experiment and dynamic system modelling.

In the first step of the proposed method we calculate the DFT using the values of y_m. Then we determine the weights on the basis of the approximation (5) outlined below. This method makes use of a relation between the multidimensional Fourier series decompostion and the multidimensional DFT.

The S-dimensional DFT from f_e is defined by

$$\begin{aligned} F(K) &= \sum_{u_1=0}^{N-1}\sum_{u_2=0}^{N-1}\cdots\sum_{u_S=0}^{N-1} f_e(U\cdot p) \\ &\quad \cdot e^{-j2\pi(k_1u_1+k_2u_2\cdots+k_Su_S)/N} \\ &= \sum_{u_1=0}^{N-1}\sum_{u_2=0}^{N-1}\cdots\sum_{u_S=0}^{N-1} f_e(U\cdot p)e^{-j2\pi K^TU/N}, \end{aligned}$$

where $K = [k_1, k_2, \ldots, k_S]^T$, $U = [u_1, u_2, \ldots, u_S]^T$, the components of U and K being nonnegative integers less than or equal to $N-1$.

Using the inverse discrete Fourier transform (IDFT), we can write

$$\begin{aligned} f_e(U\cdot p) &= \frac{1}{N^S}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\cdots\sum_{k_S=0}^{N-1} F(k_1,k_2,\ldots,k_s) \\ &\quad \cdot e^{j2\pi(k_1u_1+k_2u_2+\cdots+k_Su_S)/N} \\ &= \frac{1}{N^S}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\cdots\sum_{k_S=0}^{N-1} F(K)e^{j2\pi K^TU/N}. \end{aligned} \tag{2}$$

We also note that

$$\begin{aligned} f(U_R\cdot p) &\approx \frac{1}{N^S}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\cdots\sum_{k_S=0}^{N-1} F(k_1,k_2,\ldots,k_s) \\ &\quad \cdot e^{j2\pi(k_1u_{R1}+k_2u_{R2}+\cdots+k_Su_{RS})/N} \\ &= \frac{1}{N^S}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\cdots\sum_{k_S=0}^{N-1} F(K)e^{j2\pi K^TU_R/N}, \end{aligned} \tag{3}$$

where $U_R = [u_{R1}, u_{R2}, \ldots, u_{RS}]^T$ and the components of U_R are nonnegative real numbers less than or equal to $N-1$. The accuracy of the approximation (3) depends on the additive noise ε and the number N. The larger N, the better the approximation result. The approximation (3) can be rewritten as

$$\begin{aligned} f(U\cdot p) \approx r + \frac{1}{N^S}&\sum_{k_1=1}^{N/2-1}\sum_{k_2=1}^{N/2-1} \\ &\cdots\sum_{k_S=1}^{N/2-1} F(A_1)e^{j2\pi U^TA_1/N} \\ &+ F(A_2)e^{j2\pi U^TA_2/N} \\ &+\cdots+F(A_{2^S})e^{j2\pi U^TA_{2^S}/N}. \end{aligned} \tag{4}$$

In the above equation, r is the sum of all combinations of the following sums:

$$\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}\cdots\sum_{k_S=0}^{N-1}\frac{1}{N^S}F(K)e^{j2\pi U^TK/N},$$

where at least one component of the vector K is zero and where we omit the symbols of the sum corresponding to the zero components of K. The vector A_n is determined as follows:

(a) we write $n-1$ as a binary number which has S bits,

(b) we create a vector A_n on the basis of the vector K in the following manner: if the i-th bit of a binary number is not zero, then the i-th component of the vector A_n is equal to $(N-k_i)$, otherwise it is equal to k_i.

For example, $S=4$, $n=4$, $n-1$ is expressed as the binary number 0011, $A_n = [k_1, k_2, N-k_3, N-k_4]^T$.

We observe that

$$\begin{aligned} &e^{-j2\pi((N-k_1)u_1+(N-k_2)u_2+(N-k_3)u_3+\cdots+(N-k_S)u_S)/N} \\ &\quad = \left(e^{-j2\pi(k_1u_1+k_2u_2+k_3u_3+\cdots+k_Su_S)/N}\right)^*, \\ &e^{-j2\pi((k_1u_1+(N-k_2)u_2+(N-k_3)u_3+\cdots+(N-k_S)u_S)/N} \\ &\quad = \left(e^{-j2\pi((N-k_1)u_1+k_2u_2+k_3u_3+\cdots+k_Su_S)/N}\right)^*, \\ &e^{-j2\pi((k_1u_1+k_2u_2+(N-k_3)u_3+\cdots+(N-k_S)u_S)/N} \\ &\quad = \left(e^{-j2\pi((N-k_1)u_1+(N-k_2)u_2+k_3u_3+\cdots+k_Su_S)/N}\right)^*, \\ &\quad \vdots. \end{aligned}$$

We also note that in the case of the DFT calculated using

real data we get

$$\begin{aligned}
&F(N-k_1,N-k_2,N-k_3,\ldots,N-k_S)\\
&\quad=F^*(k_1,k_2,k_3,\ldots,k_S),\\
&F(k_1,N-k_2,N-k_3,\ldots,N-k_S)\\
&\quad=F^*(N-k_1,k_2,k_3,\ldots,k_S),\\
&F(k_1,k_2,N-k_3,\ldots,N-k_S)\\
&\quad=F^*(N-k_1,N-k_2,k_3,\ldots,k_S),\\
&\quad\vdots.
\end{aligned}$$

It is the symmetry feature of DFT coefficients. Thus, using the two foregoing dependencies and the relationship $(ab)^*=(a^*)\,(b^*)$, we can rewrite (4) as

$$\begin{aligned}
&f(U\cdot p)\\
&\approx r\\
&+\frac{2}{N^S}\sum_{k_1=1}^{N/2-1}\sum_{k_2=1}^{N/2-1}\cdots\sum_{k_S=1}^{N/2-1}\operatorname{Re}\{F(k_1,k_2,\ldots,k_S)\}\\
&\cdot\cos\left(p(k_1u_1+k_2u_2+\cdots+k_Su_S)\right)\\
&-\operatorname{Im}\{F(k_1,k_2,\ldots,k_S)\}\\
&\cdot\sin\left(p(k_1u_1+k_2u_2+\cdots+k_Su_S)\right)\\
&+\operatorname{Re}\{F(k_1,k_2,\ldots,N-k_S)\}\\
&\cdot\cos\left(p(k_1u_1+k_2u_2+\cdots+k_{S-1}u_{S-1}-k_Su_S)\right)\\
&-\operatorname{Im}\{F(k_1,k_2,\ldots,N-k_S)\}\\
&\cdot\sin\left(p(k_1u_1+k_2u_2+\cdots+k_{S-1}u_{S-1}-k_Su_S)\right)\\
&+\cdots\\
&+\operatorname{Re}\{F(k_1,N-k_2,\ldots,N-k_S)\}\\
&\cdot\cos\left(p(k_1u_1-k_2u_2-\cdots-k_Su_S)\right)\\
&-\operatorname{Im}\{F(k_1,N-k_2,\ldots,N-k_S)\}\\
&\cdot\sin\left(p(k_1u_1-k_2u_2-\cdots-k_Su_S)\right).
\end{aligned}\tag{5}$$

The approximation (5) allows efficient calculation of the weights independently of the value S. In order to get the values of individual weights, it is sufficient to express the sine and cosine functions whose arguments are the weighted sums of several inputs in (5) as the sums of products of the sine and cosine functions whose arguments depend on individual inputs only. Next, the coefficients before identical products of functions with the same arguments have to be summed up. The weights are equal to these sums. (The sum of the coefficients preceding the product of the functions calculated by the i-th multiplying node, $i=1,\ldots,N^S$, is equal to the weight connected with this node.)

Example 1. For a two-input network, we have

$$\begin{aligned}
f(Up)\;\approx\;&r+\frac{2}{N^2}\sum_{k_1=1}^{N/2-1}\sum_{k_2=1}^{N/2-1}\left(\operatorname{Re}\{F(k_1,k_2)\}\right.\\
&\left.\cdot\cos(pk_1u_1+pk_2u_2)\right.\\
&-\operatorname{Im}\{F(k_1,k_2)\}\sin(pk_1u_1+pk_2u_2)\\
&+\operatorname{Re}\{F(k_1,N-k_2)\}\cos(pk_1u_1-pk_2u_2)\\
&-\operatorname{Im}\{F(k_1,N-k_2)\}\sin(pk_1u_1-pk_2u_2))\\
=\;&r+\frac{2}{N^2}\sum_{k_1=1}^{N/2-1}\sum_{k_2=1}^{N/2-1}(\operatorname{Re}\{F(k_1,k_2)\}\\
&+\operatorname{Re}\{F(k_1,N-k_2)\})\\
&\cdot\cos(pk_1u_1)\cos(pk_2u_2)\\
&+(-\operatorname{Re}\{F(k_1,k_2)\}+\operatorname{Re}\{F(k_1,N-k_2)\})\\
&\cdot\sin(pk_1u_1)\sin(pk_2u_2)\\
&+(-\operatorname{Im}\{F(k_1,k_2)\}-\operatorname{Im}\{F(k_1,N-k_2)\})\\
&\cdot\sin(pk_1u_1)\cos(pk_2u_2)\\
&+(-\operatorname{Im}\{F(k_1,k_2)\}+\operatorname{Im}\{F(k_1,N-k_2)\})\\
&\cdot\cos(pk_1u_1)\sin(pk_2u_2),
\end{aligned}$$

where

$$\begin{aligned}
r=\;&\frac{F(0,0)}{N^2}\\
&+\frac{2}{N^2}\sum_{k_1=1}^{N/2-1}(\operatorname{Re}\{F(k_1,0)\}\cos(pk_1u_1)\\
&-\operatorname{Im}\{F(k_1,0)\}\sin(pk_1u_1))\\
&+\frac{2}{N^2}\sum_{k_2=1}^{N/2-1}(\operatorname{Re}\{F(0,k_2)\}\cos(pk_2u_2)\\
&-\operatorname{Im}\{F(0,k_2)\}\sin(pk_2u_2)).
\end{aligned}$$

♦

Using (5), it is easy to implement a computer program which will symbolically compute the weights. To this end, it is sufficient to implement symbolic calculations for the sine and cosine functions for many arguments (for example, recurrently) and to group the coefficients preceding the products of the same harmonic functions. No iteration learning is feasible for the presented method. Additional learning of the network can be run online with, e.g., a simple gradient descent algorithm which, in the case of an ONN, ensures good results (see Fig. 2).

The described algorithm cannot be directly applied when input data are not evenly spaced. Such a situation is typical when modelling dynamic systems where the current output value depends on the previous output values. Two methods are proposed in such a case.

The first method is based on partitioning the input data hypercube into identical smaller sub-hypercubes. All input data which fall into each hypercube are averaged. These averages are used to calculate a multidimensional FFT. Due to the speed of the FFT calculations, it is recommended to choose N as a highly factorizable number. The method can be applied when we are able to select training data which fall evenly enough into these

sub-hypercubes. This method requires the training set to be sufficiently large. If the data are not evenly spaced in some dimensions only, then they can be averaged just in these dimensions. It is worth mentioning that the method ensures favourable generalization properties for the formulated model, even without network pruning. This is because the number of sub-hypercubes is much less than the number of averaged data items. The second method makes use of the property of multidimensional DFT separability. It requires modification in computations for the one-dimensional DFT in those dimensions where the data are not evenly spaced. The nonuniform fast fourier transform (NUFFT) algorithm with $O(N\log_2 N)$ complexity is especially useful here. The NUFFT is described in (Dutt and Rokhlin, 1993; Liu and Nyguen, 1998).

The methods may be combined with each other, i.e., one of them might be used for some dimensions and another for the remaining dimensions. ♦

Example 2. The example of the first method applied in modelling a nonlinear dynamic system by an FSNN is depicted in Fig. 3. The system was described by

$$y_k = 6\exp\left(-(y_{k-1}-3)^2/4-(y_{k-2}-3)^2/4\right),$$

where y_k, y_{k-1}, y_{k-2} denote the system output values at time instants k, $k-1$, $k-2$, respectively.

It was assumed that at our disposal are only the measurements of the output values disturbed by the additive white noise with zero mean and a variance of 0.25. In this example, 100,000 data items were used which were then averaged in 15×15 fields. The weights were calculated on the basis of the two-dimensional FFT of the size 15×15.

Due to the high speed of operation, the method is suitable for processing intense data streams which force continuous learning or extra learning of the network. ♦

After calculating the weights, pruning can be performed by removing some of the connections. Network generalization capabilities could be improved by means of network pruning. The model variance is reduced, but its bias increases. Therefore, attempts have to be made for an appropriate bias/variance trade-off (Nelles 2001). Let $b_i = w_i \cdot (\sqrt{\pi})^{\alpha_i} \cdot \left(\sqrt{2\pi}\right)^{\beta_i}$, $i = \{1, 2, (N-1)^S\}$, where α_i is the number of neurons that have an activation function equal to 1 and are connected with a product node, which is associated with w_i, whereas β_i is the number of neurons interconnected with the same product node and having other activation functions. The smallest mean squared error is obtained when we remove connections with smallest $|b_i|$. This stems from the fact that the trigonometric basis functions, spanning regression in $\mathbb{R}^S$, which we obtain calculating the Kronecker products within the round brackets in (1) after multiplying by $(1/\sqrt{\pi})^{\alpha_i} \cdot \left(1/\sqrt{2\pi}\right)^{\beta_i}$, are orthonormal. Therefore, to run pruning when the weight values are known, the following procedure is proposed:

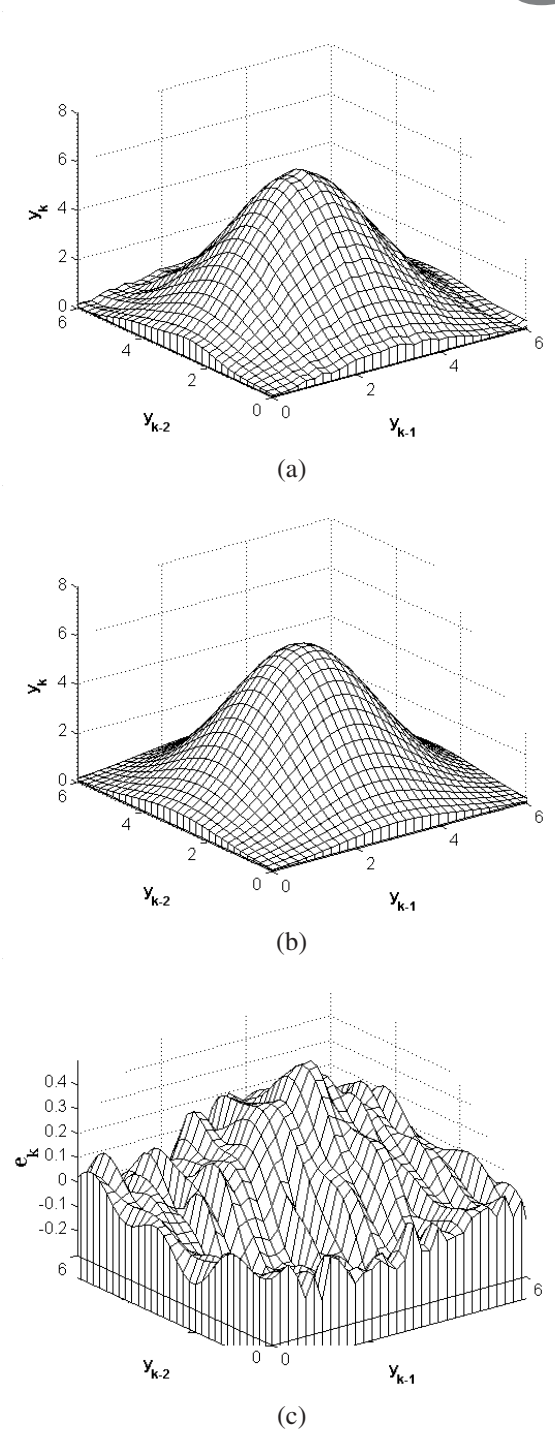

Fig. 3. First method applied for modelling a nonlinear dynamic system: (a) system output, (b) FSNN output, (c) modelling error (the system output minus the FSNN output).

Step (a): Compute $|b_i|$ for all weights.

Step (b): Sort the values of $|b_i|$.

Step (c): Remove those product nodes which are correlated with the smallest values of $|b_i|$ out of the network.

Table 1. Results of Example 3.

No. of neurons	1023	501	151	61	41
MSE	0.9776	0.9229	0.6122	0.5465	0.7342

An example illustrating how pruning affects results attained is shown below.

Example 3. An FSNN was taught to model a nonlinear SISO system whose output value was given by the function $f(x) = -0.1x^3 + 0.5 \cdot x^2 \sin(3.5x) + 2x + 2 + 8/(x+1)$, where x denotes its input. The network was taught using 1024 measurements of a network system output disturbed by additive Gaussian noise with zero mean and unit standard deviation. When learning was completed, network pruning was run. The network performance was evaluated on a testing set. Table 1 summarizes the number of neurons after pruning and the MSE reached between the network output and the true output value from the system (undisturbed by noise).

An acceptable bias/variance trade-off exists for 61 neurons. ♦

The proposed algorithm may be briefly summarized as follows:

Step 1: If the data are evenly spaced, then calculate the FFT. Otherwise, proceed according to one of the aforementioned methods.

Step 2: Derive the weights of the network using (5).

Step 3: Prune the network.

When compared with the least squares method, the way of determining the FSNN weights has several good features which include, but are not limited to, a lesser computational complexity, which is detailed in the next section. This yields a quick learning process for large networks. Then, when the weights are known, there is no problem with pruning. Therefore, good properties of network generalization can be ensured along with a good bias/variance trade-off. In the case of algorithms of a larger computational complexity, sometimes the only reasonable selection is to repeat the forward selection several times (Nelles, 2001).

3. Computational complexity

The computational complexity of the FFT algorithm for determining the N-point one-dimensional DFT is $O(N \log_2 N)$. To determine the S-dimensional DFT, it is necessary to calculate SN^{S-1} one-dimensional DFTs. The computational complexity of calculating the S-dimensional DFT is $O(SN^S \log_2 N)$. If an S-dimensional DFT is to be used to determine $(N-1)^S$ weights, 2^{S-1} real numbers will be summed for each weight. Hence the total computational complexity of the proposed method is $O(N^S \log_2 N + N^S)$. When $(N-1)^S$ weights are to be calculated directly from the inner products $w_k = \langle f_e(U), v_k(U) \rangle$, $k = 1, 2, \ldots, (N-1)^S$, this would require about $(N-1)^S N^S = (N^2-N)^S$ additions and multiplications of complex numbers.

FFT algorithms are most efficient when N is a highly factorizable number. This paper provides no description of well-known methods of handling other data sizes (Chu and George, 2000; Bracewell, 1999). Most often, the input data are purely real, in which case the DFT satisfies a symmetry condition. Efficient FFT algorithms have been designed for this situation.

4. Necessary modifications when network inputs are associated with different numbers of neurons

If the i-th input is associated with $N_i - 1$ neurons, where N_i is an even number, then instead of the approximation (5), the following approximation should be used to calculate the weights:

$$
\begin{aligned}
&f(u_1 p_1, \ldots, u_S p_S) \\
&\approx r_M + \frac{2}{\prod_{i=1}^{S} N_i} \\
&\quad \cdot \sum_{k_1=1}^{N_1/2-1} \sum_{k_2=1}^{N_2/2-1} \cdots \sum_{k_S=1}^{N_S/2-1} \operatorname{Re}\{F(k_1, k_2, \ldots, k_S)\} \\
&\quad \cdot \cos(p_1 k_1 u_1 + p_2 k_2 u_2 + \cdots + p_S k_S u_S) \\
&\quad - \operatorname{Im}\{F(k_1, k_2, \ldots, k_S)\} \\
&\quad \cdot \sin(p_1 k_1 u_1 + p_2 k_2 u_2 + \cdots + p_S k_S u_S) \\
&\quad + \operatorname{Re}\{F(k_1, k_2, \ldots, N - k_S)\} \\
&\quad \cdot \cos(p_1 k_1 u_1 + \cdots + p_{S-1} k_{S-1} u_{S-1} - p_S k_S u_S) \\
&\quad - \operatorname{Im}\{F(k_1, k_2, \ldots, N - k_S)\} \\
&\quad \cdot \sin(p_1 k_1 u_1 + \cdots + p_{S-1} k_{S-1} u_{S-1} - p_S k_S u_S) \\
&\quad + \cdots \\
&\quad + \operatorname{Re}\{F(k_1, N - k_2, \ldots, N - k_S)\} \\
&\quad \cdot \cos(p_1 k_1 u_1 - p_2 k_2 u_2 - \cdots - p_S k_S u_S) \\
&\quad - \operatorname{Im}\{F(k_1, N - k_2, \ldots, N - k_S)\} \\
&\quad \cdot \sin(p_1 k_1 u_1 - p_2 k_2 u_2 - \cdots - p_S k_S u_S). \qquad (6)
\end{aligned}
$$

In the above approximation, $p_i = 2\pi/N_i$, and r_M is the sum of all the combinations of the following sums:

$$
\sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \cdots \sum_{k_S=0}^{N_S-1} \frac{1}{\prod_{i=1}^{S} N_i} F(k_1, k_2, \ldots, k_s) \cdot e^{j(p_1 k_1 u_1 + p_2 k_2 u_2 + \cdots + p_S k_S u_S)},
$$

where at least one component of the vector K is zero and where we omit the symbols of the sum corresponding to the zero components of K.

In order to derive (6), the following equation is used:

$$f_e(p_1u_1, p_2u_2, \ldots, p_Su_S) = \frac{1}{\prod_{i=1}^{S} N_i} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \cdots \sum_{k_S=0}^{N_S-1} F(k_1, k_2, \ldots, k_S) \cdot e^{p_1k_1u_1+p_2k_2u_2+\cdots+p_Sk_Su_S}$$

instead of (2), and then the calculations described in Section 2 are applied.

5. Conclusion

The presented method of determining weights has several advantages. It is characterized by a low computational complexity. It is far more efficient than computing inner products using numerical integration. Software calculating the inverse discrete Fourier transform using FFT algorithms is widespread and easily available. Using the property of separability, we can obtain a multidimensional DFT by successive calculations of one-dimensional DFTs. The described method is especially well-suited for processing data streams which have been sampled at regular intervals. A drawback of the ONN is that the number of weights increases exponentially with the dimension of input data. In order to reduce the dimension of input data, we can try to use projection pursuit methods (Hyvarinen and Oja, 2000; Kegl *et al*., 2000; Li and Sun, 2005; Joliffe, 1986). It is worth mentioning that the FFT algorithm can be also applied for effective calculation of FSNN outputs. In such a case, at first we need to determine the coefficients of the DFT on the basis of the network weights, and then we calculate the IDFT. This method was described in detail in (Halawa, 2008). As the method has a low calculation complexity, it is especially suitable for data stream processing.

Acknowledgment

This work was supported by the Polish Ministry of Science and Higher Education under a research grant contract from 2006 to 2009.

References

Bracewell R. (1999). *The Fourier Transform and Its Applications, 3rd Edn.*, McGraw-Hill, New York, NY.

Chu E. and George A. (2000). *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, CRC Press, Boca Raton, FL.

Dutt A. and Rokhlin V. (1993). Fast Fourier transforms for nonequispaced data, *Journal of Scientific Computing* **14**(6): 1368–1393.

Gonzalez R. C. and Woods R. E. (1999). *Digital Image Processing, 2nd Edn.*, Prentice-Hall, Inc., Boston, MA.

Halawa K. (2008). Fast method for computing outputs of Fourier neutral networks, *in:* K. Malinowski and L. Rutkowski, Eds., *Control and Automation: Current Problems and Their Solutions*, EXIT, Warsaw, pp. 652–659, (in Polish).

Hyvarinen A. and Oja E. (2000). Independent component analysis: Algorithms and applications, *Neural Networks* **13**(4): 411–430.

Joliffe I. T. (1986). *Principal Component Analysis*, Springer-Verlag, New York, NY.

Kegl B., Krzyżak A., Linder T. and Zeger K. (2000). Learning and design of principal curves, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(13): 281–297.

Li H. and Sun Y. (2005). The study and test of ICA algorithms, *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*, Wuhan, China, pp. 602–605.

Liu Q. H. and Nyguen N. (1998). An accurate algorithm for nonuniform fast Fourier transforms, *Microwave and Guided Wave Letters* **8**(1): 18–20.

Nelles O. (2001). *Nonlinear System Identification: From Classical Approaches to Neural Network and Fuzzy Models*, Springer-Verlag, Berlin.

Rafajłowicz E. and Pawlak M. (1997). On function recovery by neural networks based on orthogonal expansions, *Nonlinear Analysis, Theory and Applications* **30**(3): 1343–1354.

Rafajłowicz E. and Skubalska-Rafajłowicz E. (1993). FFT in calculating nonparametric regression estimate based on trigonometric series, *International Journal of Applied Mathematics and Computer Science* **3**(4): 713–720.

Sher C. F., Tseng C. S. and Chen, C. (2001). Properties and performance of orthogonal neural network in function approximation, *International Journal of Intelligent Systems* **16**(12): 1377–1392.

Tseng C. S. and Chen C. S. (2004). Performance comparison between the training method and the numerical method of the orthogonal neural network in function approximation, *International Journal of Intelligent Systems* **19**(12): 1257–1275.

Van Loan C. (1992). *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, PA.

Walker J. (1996). *Fast Fourier Transforms*, CRC Press, Boca Raton, FL.

Zhu C., Shukla D. and Paul, F. (2002). Orthogonal functions for system identification and control, *in:* C.T. Leondes (Ed.), *Neural Networks Systems, Techniques and Apllications: Control and Dynamic Systems,* Academic Press, San Diego, CA, pp. 1–73.

Received: 15 January 2007
Revised: 16 April 2007
Re-revised: 14 May 2008