LETTER Communicated by Andrzej Cichocki

A Homomorphic Neural Network for Modeling and Prediction

Maciej Pedzisz
m.pedzisz@imperial.ac.uk
Danilo P. Mandic
d.mandic@imperial.ac.uk
Imperial College London, Department of Electrical and Electronic Engineering, Communications and Signal Processing Group, London SW7 2AZ, U.K.

A homomorphic feedforward network (HFFN) for nonlinear adaptive filtering is introduced. This is achieved by a two-layer feedforward architecture with an exponential hidden layer and logarithmic preprocessing step. This way, the overall input-output relationship can be seen as a generalized Volterra model, or as a bank of homomorphic filters. Gradient-based learning for this architecture is introduced, together with some practical issues related to the choice of optimal learning parameters and weight initialization. The performance and convergence speed are verified by analysis and extensive simulations. For rigor, the simulations are conducted on artificial and real-life data, and the performances are compared against those obtained by a sigmoidal feedforward network (FFN) with identical topology. The proposed HFFN proved to be a viable alternative to FFNs, especially in the critical case of online learning on small- and medium-scale data sets.

1 Introduction

Classical system identification, signal modeling, and prediction have been traditionally dominated by linear autoregressive moving average (ARMA) models (Söderström & Stoica, 1994), despite the real-world problems being typically nonlinear and nongaussian. One step toward regression based on nonlinear modeling is to use artificial neural networks with nonlinear activation functions (AF)—nonlinear ARMA (NARMA) models (Mandic & Chambers, 2001)—which considerably enhances modeling capabilities.

Based on their topology, there are two main types of such networks: feedforward networks (FFNs) (Haykin, 1994) and recurrent neural networks (RNNs) (Mandic & Chambers, 2001). From a mathematical viewpoint, the former represent static nonlinear maps, while the latter are nonlinear dynamic feedback systems (Narendra & Parthasarathy, 1990). In practice, both models can be used interchangeably, depending on available resources, that

is, a problem that is successfully solved using a large-scale FFN can be equally solved using a smaller-scale RNN (Mandic & Chambers, 2001).

For either type of network, a common choice of AF is a sigmoid or hyperbolic tangent function. This choice is based on the Kolmogorov universal approximation theorem (Kolmogorov, 1957) and its improvements and simplifications (Doss, 1976; Lippman, 1987; Cybenko, 1989; Funahashi, 1989; Hornik, 1990). Restricting the choice of AFs to bounded, monotonically increasing, and differentiable functions makes it possible to obtain a smooth approximation of an arbitrary input-output mapping. Indeed, many applications based on adaptive NARMA models make use of the potential of this approach (see equations 3.1 and 3.2 in section 3 and Mandic & Chambers, 2001, for details).

Notice, however, that from the system theory viewpoint, NARMA models, despite their widespread use, represent only a narrow class of nonlinear models. Indeed, Volterra models (which can be seen as a Taylor series expansion with memory) are capable of general nonlinear approximation (see equation 3.5 for a detailed example). They are well understood, but despite their theoretical advantages, they also suffer from high sensitivity and computational burden. In addition, the Volterra kernels need to be bounded and differentiable, and the powers in the expansion are defined over $\mathbb{N}$, which limits their practical application.

It is therefore natural to ask whether it is possible to introduce a neural model (and therefore inherit the generalization advantages of NNs) that will also be able to model Volterra-like systems. To that end, we introduce a class of homomorphic feedforward networks (HFFNs) that have architecture similar to that of FFNs but a radically different nonlinear input-output mapping.

By design, the proposed HFFN-based networks have the following advantages: (1) they are capable of representing general Volterra models; (2) the modeling capabilities reach beyond the Volterra type of approximation (e.g., the powers in the expansion are defined over $\mathbb{R}$); (3) the weights within such a network have unique physical interpretation; and (4) the architecture can be analyzed within the homomorphic filtering framework (filtering of multiplied signals; Oppenheim, Schafer, & Stockham, 1968).

Our aim is to introduce HFFNs for temporal problems, and hence the architecture and learning algorithms are derived for HFFN serving as a nonlinear adaptive filter. Extensions of the proposed approach to other classes of problems are straightforward.

This letter is organized as follows. In section 2 we present the HFFN architecture along with associated gradient-based training. Similarities and differences as compared to standard FFN are highlighted. The underlying modeling capabilities, relationship to FFN, and Volterra-like systems are addressed in section 3. In section 4 we illustrate the operation of the HFFN from a homomorphic filtering point of view. The choices of learning parameters and initial weights, as well as performance and learning speed,

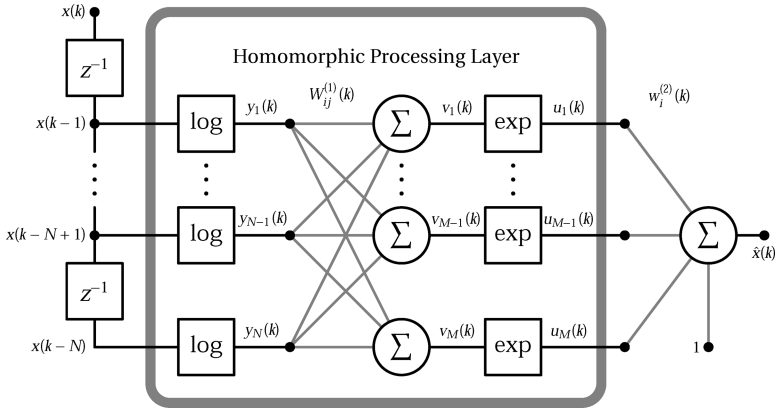

Figure 1: Homomorphic feedforward network.

are assessed by computer simulations on artificial and real-life signals. The results of these experiments are provided in section 5. Concluding remarks are presented in section 6.

2 The HFFN Architecture

The two-layer HFFN architecture is shown in Figure 1, with the homomorphic processing layer highlighted within the gray frame.

2.1 Signal Flow Within HFFN. The input to the structure (network) at time instant k is the output of a tapped delay line (TDL) fed by a real and positive signal $x(k)$.[1] In vector notation, this can be represented by a delay vector $\mathbf{x}(k)$ composed of N successive signal values, given by

$$\mathbf{x}(k) = [x(k-1),\ x(k-2),\ \ldots,\ x(k-N)]^{\mathrm{T}}. \tag{2.1}$$

This tapped input line is next passed through a logarithmic function to obtain

$$\mathbf{y}(k) = \log(\mathbf{x}(k)) = [y_1(k),\ y_2(k),\ \ldots,\ y_N(k)]^{\mathrm{T}}, \tag{2.2}$$

[1]Since normalization of the input signals is a common practice in neural computation, system identification, and control, the assumption $x(k) > 0$ does not affect the generality of the proposed approach. For online learning methodologies, where an ordinary shift is not adequate, we can apply a sigmoidal function to the input and its inverse to the network output. In such an approach, slope and range of the sigmoidal function can be adaptive in order to ensure that the input signal is positive and bounded. On the other hand, the particular choice of these two parameters can affect the results, and this issue requires further investigation.

which forms the input for the first layer of HFFN, for which the corresponding weights are denoted by $W_{ij}^{(1)}(k)$. The resulting intermediate signal $v_i(k)$ can be expressed as

$$v_i(k) = \sum_{j=1}^{N} W_{ij}^{(1)}(k) y_j(k), \quad 1 \le i \le M \quad \Longleftrightarrow \quad \mathbf{v}(k) = \mathbf{W}^{(1)}(k)\mathbf{y}(k), \tag{2.3}$$

where M is the number of elements (neurons) in the layer, $\mathbf{v}(k)$ is the output of the linear part of the first layer, and $\mathbf{W}^{(1)}(k)$ is an $M \times N$ matrix comprising the weights representing the first layer. The resulting linear combination $\mathbf{v}(k)$ is then passed through an exponential function (nonlinear activation function) to form the input $u_i(k)$ to the second (linear) layer,

$$u_i(k) = \begin{cases} \exp(v_i(k)), & 1 \le i \le M \\ 1, & i = M+1 \end{cases} \quad \Longleftrightarrow \quad \mathbf{u}(k) = \begin{bmatrix} \exp(\mathbf{v}(k)) \\ 1 \end{bmatrix}, \tag{2.4}$$

where an additional $(M+1)$th element is set to unity and represents bias. Finally, the weights $w_i^{(2)}(k)$ of the second layer scale $u_i(k)$ produce an estimator (predictor) $\hat{x}(k)$ of the original signal $x(k)$, given by

$$\hat{x}(k) = \sum_{i=1}^{M+1} w_i^{(2)}(k) u_i(k) = \mathbf{u}^{\mathrm{T}}(k)\mathbf{w}^{(2)}(k), \tag{2.5}$$

where $\mathbf{w}^{(2)}(k)$ denotes an $(M+1) \times 1$ vector representing the weights of the second layer and the bias term.

2.2 Gradient-Based Learning. For online prediction tasks, in general we seek a nonlinear mapping F, such that

$$\hat{x}(k) = F(x(k-1),\ x(k-2),\ \ldots,\ x(k-N)), \tag{2.6}$$

by which some cost function $E(k)$ is minimized—typically the square of the instantaneous error $e(k)$,

$$E(k) = \frac{1}{2}e^2(k), \quad \text{where} \quad e(k) = x(k) - \hat{x}(k). \tag{2.7}$$

In gradient-based learning, the aim is to update the weights iteratively so that $E(k)$ is reduced at each time step k. In such a case, a general weight

update $\Delta w(k)$ can be derived from

$$\Delta w(k) = -\eta(k)\frac{\partial E(k)}{\partial w(k)} = -\eta(k)e(k)\frac{\partial e(k)}{\partial w(k)} = \eta(k)e(k)\frac{\partial \hat{x}(k)}{\partial w(k)}, \tag{2.8}$$

where $\eta(k)$ is the learning rate, a small, positive variable.

For the structure depicted in Figure 1, the gradient-based weight adaptation can be expressed as

$$\Delta w_i^{(2)}(k) = \mu e(k)u_i(k), \quad 1 \le i \le M+1 \tag{2.9}$$

$$\Delta W_{ij}^{(1)}(k) = \nu e(k)w_i^{(2)}(k)u_i(k)y_j(k), \quad 1 \le i \le M, \quad 1 \le j \le N, \tag{2.10}$$

where ν and μ are, respectively, the learning rates of the first and second layers (assumed constant during learning). In matrix notation, these updates become

$$\mathbf{w}^{(2)}(k+1) = \mathbf{w}^{(2)}(k) + \mu e(k)\mathbf{u}(k) \tag{2.11}$$

$$\mathbf{W}^{(1)}(k+1) = \mathbf{W}^{(1)}(k) + \nu e(k)\mathbf{Z}(k)\mathbf{Y}(k), \tag{2.12}$$

where

$$\mathbf{Y}(k) = \operatorname{diag}(\mathbf{y}(k)), \quad \mathbf{Z}(k) = [\underbrace{\mathbf{z}(k),\ \mathbf{z}(k),\ \ldots,\ \mathbf{z}(k)}_{\text{N times}}]^{\mathrm{T}},$$

$$\mathbf{z}(k) = \mathbf{u}_M(k) \otimes \mathbf{w}_M^{(2)}(k), \tag{2.13}$$

and $\mathbf{u}_M(k)$ and $\mathbf{w}_M^{(2)}(k)$ are vectors obtained from the first M elements of $\mathbf{u}(k)$ and $\mathbf{w}^{(2)}(k)$, and the $\otimes$ denotes the Kronecker product (vector/matrix element-wise multiplication).

It should be noted that these results can be also derived from the standard backpropagation (BP) algorithm (Rumelhart, Hinton, & Williams, 1986). Since the BP method was originally derived for multi-input multi-output networks, the extension of HFFN to such a case is straightforward.

2.3 Similarities Between HFFNs and FFNs. If we consider the logarithm functions along with normalization and shift (ensuring that $x(k) > 0$) within HFFN as a preprocessing step, the HFFN architecture can be seen as a two-layer FFN with an exponential hidden layer and a linear output layer.

2.3.1 Gradient-Based Learning. It is straightforward to see that any gradient-based learning strategy that does not rest on the saturating property of the AF can be applied to HFFNs; this includes the momentum term

(Rumelhart et al., 1986), adaptive learning rates (Mathews & Xie, 1993), delta-bar-delta (Jacobs, 1988), the modified error (Van Ooyen & Nienhuis, 1992), and Quick Prop (Fahlman, 1988). The weights can be updated using online adaptation or batch training methodologies (Haykin, 1994). Any performance measure used for FFN can be equally applied for HFFN.

2.3.2 The Interchangeability of Learning Rate, Gain, and Weights. Thimm, Moerland, and Fiesler (1996) established a relationship among learning rate, gain, and weights in feedforward networks. These results can be extended to the case of an HFFN by introducing a gain β to the exponential AF as

$$\exp(x) \quad \longrightarrow \quad \exp(\beta x) \quad \Longleftrightarrow \quad \exp(x) = \exp(\beta x)|_{\beta=1}. \tag{2.14}$$

For such a function, the following relation holds (Thimm et al., 1996)[2]:

$$\frac{\beta}{\beta'} = \sqrt{\frac{\eta'}{\eta}} = \frac{\mathbf{w}'}{\mathbf{w}}. \tag{2.15}$$

In other words, two neural networks *NN* and *NN*′ that differ only in their learning rates η and η', their weights $\mathbf{w}'$ and $\mathbf{w}$, and the gain of the activation function β and β' behave in the same way. Based on this result, it is clear that the variations in the gain can be compensated by adequate changes in the learning rate and weight variance (Thimm & Fiesler, 1997b). This also means that in the case of HFFN, introducing slope β into a activation function cannot be justified from a theoretical viewpoint.

2.4 Differences Between HFFNs and FFNs. Since the two architectures differ in data preprocessing steps as well as in the properties of the AFs, not all the features of a standard sigmoidal FFN are shared by HFFN.

2.4.1 Modeling Capabilities. Unlike standard FFN, the proposed HFFN is not limited to representing NARMA mappings, as discussed in section 3.

2.4.2 Multilayer Networks. Extension of HFFN to a multilayer network is not as straightforward as in the case of an ordinary FFN. For illustration, consider replicating the hidden layer (marked with a gray frame in Figure 1) to form a multilayered network. Knowing that the link between two consecutive layers becomes a concatenation of logarithmic and exponential functions, all hidden layers but the last one degenerate into a linear layer. In such a case, the whole network can be replaced by a standard HFFN with

[2]The proof is valid for any differentiable nonlinear AF with the first derivative that satisfies (Mandic & Chambers, 2001): $\beta \frac{\partial \Phi(\beta,x)}{\partial(\beta x)} = \frac{\partial \Phi(\beta,x)}{\partial x}$.

one exponential hidden layer and a linear output layer. This also illustrates the compactness of the mappings performed by HFFNs (see section 3 for more detail).

2.4.3 Expansive Character of the AF. Another important difference between HFFNs and FFNs is that the HFFN cannot be trained using standard normalized (or a posteriori) algorithms. Following Mandic and Chambers (2000a), deriving such an algorithm from the Taylor series expansion of the output error would lead to a condition fulfilled only by contractive AFs (see Mandic & Chambers, 2000b; Mandic, Hanna, & Razaz, 2001; or Hanna & Mandic, 2003, for more detail), which is not the case with the exponential AFs.

2.4.4 Algorithm Initialization. Notice that for a layer with "squashing" (sigmoidal or hard-limiting) AFs, choosing too big a learning rate will not lead to divergence: the outputs will be saturated and bounded. Due to the expansive behavior of the exponential AF within HFFN, standard settings for learning rates and initial weights variances (Thimm & Fiesler, 1997a) may not be applicable. For an exponential AF within an HFFN, too large a learning rate will lead to an unbounded output, and thus instability.

2.4.5 Computational Overhead. For a standard FFN, we need to calculate one exponential, one multiplication, one division, and one summation per neuron (see equation 3.1). In the case of HFFN, these calculations reduce to one exponential. This gain in calculation speed is particularly visible when the network is large scale or batch-trained on large data sets. On the other hand, N logarithms are required in the preprocessing step. Since those are calculated only once for the entire sequence (not required for training), this computational overhead can be neglected (for batch training).

3 Approximation Capabilities of HFFNs

To ascertain whether an application based on a two-layer FFN can be implemented using an HFFN, it is necessary to compare their approximation capabilities. To that end, let us perform a Taylor series expansion of the sigmoidal AF of a single neuron (Billings, Jamaluddin, & Chen, 1992):

$$\Phi(v(k)) = \frac{1}{1+\mathrm{e}^{-\beta v(k)}} = \frac{1}{2} + \frac{\beta}{4}v(k) - \frac{\beta^3}{48}v^3(k) + \frac{\beta^5}{480}v^5(k) - \frac{17\beta^7}{80640}v^7(k) + \cdots \tag{3.1}$$

This clearly illustrates the complex nonlinear modeling capabilities of those networks. But such complex behavior is not always desirable. To show this,

let us consider a simple AR model described by

$$x(k) = 1 - \frac{1}{4}x(k-1). \tag{3.2}$$

Since there is only one previous sample on which $x(k)$ depends (memory of order 1), we can take only one sigmoidal neuron with a single weight a and a bias b. Fixing $\beta = 1$ and taking into account the first three terms in the expansion 3.1, we have

$$\begin{aligned}\hat{x}(k) &= \Phi(a\,x(k-1)+b) \approx \frac{1}{2} + \frac{1}{4}(a\,x(k-1)+b) - \frac{1}{48}(a\,x(k-1)+b)^3 \\ &= \left(\frac{1}{2} + \frac{b}{4} - \frac{b^3}{48}\right) + \left(\frac{a}{4} - \frac{3ab^2}{48}\right)x(k-1) - \frac{3a^2b}{48}x^2(k-1) \\ &\quad - \frac{a^3}{48}x^3(k-1).\end{aligned} \tag{3.3}$$

First, observe that there are higher-order terms that do not exist in the original expression, equation 3.2. Second, the coefficient associated with $x(k-1)$ depends on both adaptable parameters. In general, by introducing more neurons, we can only decrease the influence of higher-order terms as well as dependence among weights (Haykin, 1994). These higher-order terms cannot be completely eliminated since a linear AF can be approximated by a sigmoidal one (or a linear combination of them) only in some limited range. Additionally, adding more neurons will introduce redundancy.

Let us now illustrate the approximation capabilities of an HFFN. Rewriting equation 2.5, we have

$$\begin{aligned}\hat{x}(k) &= w_{M+1}^{(2)}(k) + \sum_{i=1}^{M} w_i^{(2)}(k) \exp\left(\sum_{j=1}^{N} W_{ij}^{(1)}(k) \log\left(x(k-j)\right)\right) \\ &= w_{M+1}^{(2)}(k) + \sum_{i=1}^{M} w_i^{(2)}(k) \exp\left(\sum_{j=1}^{N} \log\left(x^{W_{ij}^{(1)}(k)}(k-j)\right)\right) \\ &= w_{M+1}^{(2)}(k) + \sum_{i=1}^{M} w_i^{(2)}(k) \prod_{j=1}^{N} x^{W_{ij}^{(1)}(k)}(k-j).\end{aligned} \tag{3.4}$$

Comparing this model to a standard FFN, we may state that for HFFNs:

- Adaptable parameters (weights) are independent of one another.
- There are no spurious higher-order terms in the representation, equation 3.4.

- All the weights have unique physical interpretation: weights $W_{ij}^{(1)}(k)$ represent the powers of $x(k-j)$ and $w_i^{(2)}(k)$ a linear combiner at the output layer.
- Weights $W_{ij}^{(1)}(k)$ (powers) are defined over $\mathbb{R}$. This is a unique feature that is not present in other neural network approaches, where $W_{ij}^{(1)}(k) \in \mathbb{N}$.

From equation 3.4, we can also observe the following properties of HFFNs:

- **Volterra-like modeling capabilities.** Imposing constraints on $W_{ij}^{(1)}(k) \in \mathbb{N}^+$, makes the HFFN an exact solution for a general class of Volterra-like filtering problems (Mathews, 1991). This can be illustrated as follows. Consider a simple second-order Volterra model with embedding dimension (Takens, 1981) equal to two, given by

$$\begin{aligned}\hat{x}(k) &= h_0(k) + \sum_i h_i(k)x(k-i) + \sum_i \sum_j g_{ij}(k)x(k-i)x(k-j) \\ &= h_0(k) + h_1(k)x(k-1) + h_2(k)x(k-2) + g_{11}(k)x^2(k-1) \\ &\quad + g_{22}(k)x^2(k-2) + 2g_{12}(k)x(k-1)x(k-2). \end{aligned} \tag{3.5}$$

  Such a model can be exactly represented by a five-neuron HFFN with weights

$$\mathbf{W}^{(1)}(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{w}^{(2)}(k) = \begin{bmatrix} h_1(k) \\ h_2(k) \\ g_{11}(k) \\ g_{22}(k) \\ 2g_{12}(k) \end{bmatrix}, \quad \text{and} \quad w_6^{(2)}(k) = h_0(k). \tag{3.6}$$

  However, since HFFNs naturally exhibit much more complex behavior ($W_{ij}^{(1)}(k) \in \mathbb{R}$), it is not justifiable to impose constraints during learning (except when an HFFN is used to identify or predict an a priori known Volterra-like system).
- **Saturation-like nonlinearities.** Since there are no extra higher-order terms that do not result from the original signal, the HFFN model cannot match exactly the equivalent FFN structure. In particular, any saturation-like nonlinearities that are naturally modeled by FFN with squashing AFs cannot be correctly represented by an HFFN with expansive, unbounded AFs.

- **Weight initialization.** Knowing that $\mathbf{W}^{(1)}(k)$ represent the powers in the expansion model as shown in equations 3.4 and 3.6, we can define an appropriate method for weight initialization,

$$\mathbf{W}^{(1)}(0) = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}, \tag{3.7}$$

where $\mathbf{1}$ is an $N \times N$ identity matrix and $\mathbf{0}$ is an $(M - N) \times N$ null matrix. Such an initialization has one main advantage: the learning process starts from the linear model, that is the unity values on the diagonal imply that input signals are not modified by an exponential layer and zeros outside the diagonal that there are no cross-terms (e.g., $x(k-i)x(k-j)$) at the output. Such a property is not present in other FFNs.

4 Homomorphic Filtering Based on HFFN

According to the generalized linear filtering theorem presented in Oppenheim et al. (1968), for two signals $s_1(t)$ and $s_2(t)$ that have been combined according to some rule denoted by $\circ$, the resulting signal $s(t)$ can be expressed as

$$s(t) = s_1(t) \circ s_2(t). \tag{4.1}$$

Then a generalized transformation ϕ for the filter needs to satisfy the following properties,

$$\phi[s_1(t) \circ s_2(t)] = \phi[s_1(t)] \circ \phi[s_1(t)], \quad \text{and} \quad \phi[c : s(t)] = c : \phi[s(t)], \tag{4.2}$$

where $\circ$ corresponds to vector addition, : to scalar multiplication, and ϕ to algebraically linear transformation in a vector space. For such a system, ϕ can be represented as a cascade of three systems (Oppenheim, 1965),

$$s(t) \longrightarrow \boxed{A_o \longrightarrow L \longrightarrow A_o^{-1}} \longrightarrow g(t) \tag{4.3}$$

where L is a linear system, and A_o is referred as the characteristic system and A_o^{-1} as its inverse. In this representation, the characteristic system A_o has the properties

$$A_o[s_1(t) \circ s_2(t)] = A_o[s_1(t)] + A_o[s_2(t)], \quad \text{and} \quad A_o[c : s(t)] = c\,A_o[s(t)]. \tag{4.4}$$

On the basis of this canonical representation, a generalized linear filter corresponds to transforming the original problem into the one in which the components are added, and after linear filtering, the result is transformed back to the original input space.

A straightforward application of this theorem is the homomorphic filtering of multiplied signals. Using the above notation, we have

$$s_1(t) \circ s_2(t) = s_1(t) \cdot s_2(t), \quad \text{and} \quad c : s(t) = s^c(t), \tag{4.5}$$

whereas system A_o needs to satisfy

$$A_o[s_1(t) \cdot s_2(t)] = A_o[s_1(t)] + A_o[s_2(t)] \tag{4.6}$$

$$A_o[s^c(t)] = c\, A_o[s(t)] \tag{4.7}$$

$$A_o^{-1}[A_o[s(t)]] = s(t). \tag{4.8}$$

If we limit ourselves to only positive real signals $s(t)$ and scalars c, the characteristic system A_o and its inverse A_o^{-1} may be chosen as the ordinary logarithm and exponential functions.

To employ this within the HFFN framework, consider the output $u_i(k)$ of the ith neuron of the exponential layer, given by

$$u_i(k) = \exp\left(\sum_{j=1}^{N} W_{ij}^{(1)}(k) \log[x(k-j)]\right). \tag{4.9}$$

Comparing this with equation 4.3, it is clear that $A_o = \log$, $A_o^{-1} = \exp$, and a linear system L corresponds to an ordinary finite impulse response (FIR) filter with its coefficients equal to $W_{ij}^{(1)}(k)$. From this point of view, the entire HFFN network can be expressed as

$$\hat{x}(k) = \sum_{i=1}^{M+1} w_i^{(2)}(k) u_i(k), \tag{4.10}$$

which can be seen as a parallel implementation (linear combination) of a bank of homomorphic filters.

A comprehensive introduction to homomorphic processing and cepstral analysis with its applications can be found in Oppenheim and Schafer (1975) and Oppenheim et al. (1968). Applications of homomorphic filtering to speech processing (vocal tract transfer characteristics) can be found in Oppenheim and Schafer (1968), Oppenheim (1969), Schafer and Rabiner (1970), and Oppenheim, Kopec, and Tribolet (1976) and for seismic data processing (isolating the impulse response of the earth's crust) in Ulrych (1971), and Stoffa, Buhl, and Bryan (1974a, 1974b).

The more recent advances in homomorphic signal processing can be found in Lindquist and Mukherjee (1994; signal companding), Marenco and Madisetti (1996; deconvolution of bandpass signals), Sabry-Rizk, Zgallai, Hardiman, and O'Riordan (1995; analysis of abnormalities in ECG signals),

Ryu, Lee, and Kwon (2004; change detection in high-resolution imagery), and Kuribayashi and Tanaka (2005; image fingerprinting).

5 Experiments

For rigor, simulations were performed on a 4- and 16-neuron HFFN for various modeling scenarios. The main objective of these experiments was to provide insight into the settings of the optimal learning rates and weight initialization, as well as to assess the overall performance and speed of convergence. The choice of the number of neurons was arbitrary and suggested by "smaller nets perform better" (Elsken, 1990).

Three implementation scenarios were considered:

- **Batch:** Batch training for 10,000 epochs; 295 samples (learning set: 150 samples, validation set: 145 samples). This scenario corresponds to the case where the number of available samples is very limited (e.g., sunspot data) and the objective is to find the best (in terms of the prediction gain) data model.
- **Online 1:** Online adaptation; 1000 samples (learning set: 800 samples, validation set: 200 samples). This online scenario corresponds to situations where the number of samples is relatively large, required processing time is small, and prediction gain should be close to the one obtained by batch training.
- **Online 2:** Online adaptation; 295 samples (learning set: 150 samples, validation set: 145 samples). This online scenario corresponds to real-time learning based on a small-scale data set. The objective is twofold: fast training and satisfactory performance.

As a measure of performance, we used a prediction gain r_p given by Haykin and Li (1995),

$$r_p \triangleq 10 \log \left(R_p\right) [\text{dB}], \quad R_p \triangleq \frac{\sigma_{\hat{x}}^2}{\sigma_e^2}, \tag{5.1}$$

where $\sigma_{\hat{x}}^2$ denotes the variance of the predicted signal $\hat{x}(k)$ and σ_e^2 the variance of the instantaneous prediction error $e(k)$.

5.1 Data Sets. Comprehensive simulations were performed on a number of artificial and real-life signals:

- **Linear AR(4) model (ar4).** Linear model proposed in Mandic (2004, first experiment, p. 116) and described by

$$\begin{aligned} y(k) = {} & 1.79y(k-1) - 1.85y(k-2) + 1.27y(k-3) \\ & - 0.41y(k-4) + x(k), \end{aligned} \tag{5.2}$$

  where $x(k)$ is white gaussian noise with variance $\sigma^2 = 1$.

- **Volterra nonlinear model I (volterra1).** This is a Volterra quadratic model described by

$$y(k) = 0.8y(k-1) + 0.2y(k-1)y(k-2) - 0.3y^2(k-3) + x(k), \quad (5.3)$$

where $x(k)$ is white gaussian noise with $\sigma^2 = 0.01$.
- **Volterra Nonlinear model II (volterra2).** This is a Volterra quadratic model proposed in Mathews (1991, equation 14 on p. 15). It is described by

$$\mathbf{h} = [-0.78,\ -1.48,\ -1.39,\ 0.04]^{\mathrm{T}}, \quad \text{linear part} \quad (5.4)$$

$$\mathbf{g} = \begin{bmatrix} 0.54 & 3.72 & 1.86 & -0.76 \\ 3.72 & -1.62 & 0.76 & -0.12 \\ 1.86 & 0.76 & 1.41 & -1.52 \\ -0.76 & -0.12 & -1.52 & -0.13 \end{bmatrix}, \quad \text{quadratic part} \quad (5.5)$$

where the input signal was obtained by processing white gaussian noise with $\sigma^2 = 0.01$ with a linear filter for which the impulse response is given by [0.25, 1.0, 0.25].
- **Nonlinear model I (non1).** This is a nonlinear system proposed in Narendra and Parthasarathy (1990, example 2 on p. 15). The system is described by

$$y(k) = \frac{y(k-1)y(k-2)[y(k-1)+2.5]}{1+y^2(k-1)+y^2(k-2)} + x(k-1), \quad (5.6)$$

where $x(k)$ is white gaussian noise with $\sigma^2 = 1$.
- **Nonlinear model II (non2).** This is a nonlinear system proposed in Narendra and Parthasarathy (1990, example 3 on p. 16). Signal model is governed by

$$y(k) = \frac{y(k-1)}{1+y^2(k-1)} + x^3(k-1), \quad (5.7)$$

where $x(k)$ is white gaussian noise with $\sigma^2 = 1$.
- **van der Pol equation (vdpol).** This is an ordinary differential equation describing self-sustaining oscillations. This equation arises in the study of circuits containing vacuum tubes and is given by Wiggins (1990)

$$y'' - \mu(1-y^2)y' + y = 0. \quad (5.8)$$

For simulations, we used $\mu = 5$.
- **Lorenz attractor (lorenz).** Lorenz attractor is a chaotic map that shows how the state of a dynamical system evolves over time in a complex, nonrepeating pattern. This system is nonlinear, three-dimensional,

and deterministic, and is described by coupled equations (Lorenz, 1963),

$$\frac{dx}{dt} = \sigma(y - x), \qquad \frac{dy}{dt} = x(\rho - z) - y, \qquad \frac{dz}{dt} = xy - \beta z, \quad (5.9)$$

with (usually) $\sigma = 10$, $\beta = 8/3$, $\rho = 28$. For simulations, we have used only the first coordinate of the Lorenz attractor.

- **Hair Dryer (dryer)** (Matlab, System Identification Toolbox). This series contains data collected from a laboratory scale hairdryer. The input is the power over the heating device; the output (time series to predict) is the outlet air temperature.
- **Handel's Hallelujah (handel)** (Matlab, Filter Design Toolbox). This is a short audio clip from Handel's Hallelujah chorus. There are 73,113 samples, sampled at $F_s = 8192$ Hz.
- **Loma Prieta Earthquake (loma)** (Matlab, Demos:Mathematics). The data set contains three variables comprising time traces from an accelerometer in the Natural Sciences building at the University of California, Santa Cruz. The accelerometer recorded the main shock of the Loma Prieta earthquake in the Santa Cruz Mountains. As a signal to predict, we took the measurement of the vertical acceleration.
- **Sunspots Activity (sunspots)** (Matlab, Demos:Mathematics). This series contains the variations in sunspot activity from 1700 to 1994. This series is cyclical, reaching a maximum about every 11 years. There are only 295 data samples collected.
- **Wind Speed (wind)** The samples used in the experiments were obtained from the Iowa Department of Transportation (publicly available from http://mesonet.agron.iastate.edu/request/awos/1min.php) and contain data acquired every minute from AWOS (Automated Weather Observing System) sensors. We chose Storm Lake (SLB) station, and the gathered data correspond to the wind speed observed in 2005.

5.2 Choice of Learning Rates. Since the standard FFN settings cannot be used directly for HFFNs (see the differences set out in section 2.4), we started our simulations by determining the optimal learning rates. For the three scenarios considered and both HFFN networks (4 and 16 neurons), we used gradient-based training and estimated prediction gain (on the validation set) as a function of the learning rates ν (first nonlinear layer) and μ (second linear layer). Weights of the nonlinear layer were initialized as described in section 3 (diagonal initialization), and those of a linear layer were all set to zero.

Simulation results for the handel data set signal are visualized in Figure 2, where darker areas of the contour plots correspond to larger prediction gains (R_p) and checked areas to conditions where the networks diverged. These plots are fairly consistent for all the signals considered, and we can

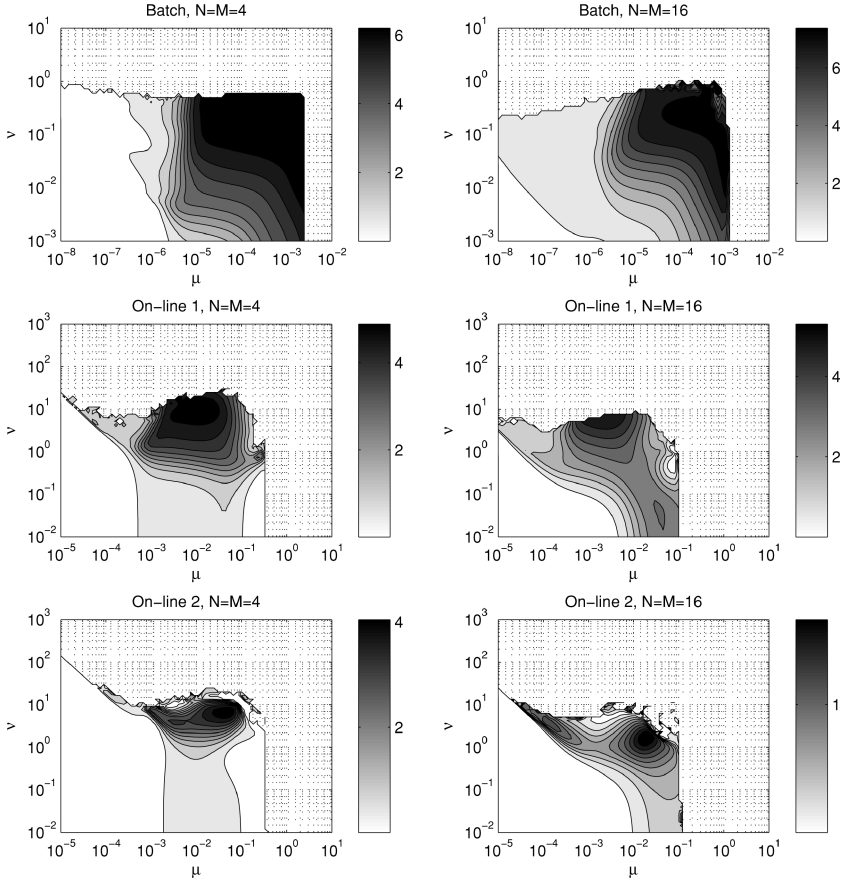

Figure 2: Prediction gain as a function of learning rates for "handel" signal.

observe that there are limit values for ν and μ beyond which the HFFN exhibits unstable behavior (in contrast to FFN with sigmoidal AFs where ν can be chosen to any value without provoking divergence), an increase in the number of neurons implies a reduction in the range of optimal learning rates, and a decrease in the number of available samples or the training time (online adaptation instead of batch training) narrows the ranges of optimal learning rates and decreases the prediction gain.

Since there is no universal choice of learning rates, the shape of $R_p(\mu, \nu)$, location, and the value of the maximum R_p depend on the data set in hand, number of neurons, and type of training. Based on this observation, for each scenario considered and all signals, we found the pairs of optimal learning rates and used them in the following experiments.

Table 1: Minimum and Maximum of the Calculated Optimal Learning Rate for the Three Scenarios and All Signals.

	$N = M = 4$		$N = M = 16$	
Batch training	$\mu_{\min} = 2.23 \cdot 10^{-8}$	$\mu_{\max} = 1.35 \cdot 10^{-3}$	$\mu_{\min} = 1.82 \cdot 10^{-8}$	$\mu_{\max} = 6.06 \cdot 10^{-4}$
	$\nu_{\min} = 7.91 \cdot 10^{-3}$	$\nu_{\max} = 7.20 \cdot 10^{-1}$	$\nu_{\min} = 1.15 \cdot 10^{-2}$	$\nu_{\max} = 4.09 \cdot 10^{-1}$
Online 1	$\mu_{\min} = 2.72 \cdot 10^{-5}$	$\mu_{\max} = 3.67 \cdot 10^{-2}$	$\mu_{\min} = 1.49 \cdot 10^{-5}$	$\mu_{\max} = 1.65 \cdot 10^{-2}$
adaptation	$\nu_{\min} = 2.36$	$\nu_{\max} = 29.8$	$\nu_{\min} = 1.94$	$\nu_{\max} = 24.5$
Online 2	$\mu_{\min} = 1.65 \cdot 10^{-4}$	$\mu_{\max} = 4.04 \cdot 10^{-1}$	$\mu_{\min} = 2.72 \cdot 10^{-5}$	$\mu_{\max} = 2.02 \cdot 10^{-2}$
adaptation	$\nu_{\min} = 0.18$	$\nu_{\max} = 16.6$	$\nu_{\min} = 1.08$	$\nu_{\max} = 36.3$

For the three scenarios, the minimal and maximal values (among all signals) of the optimal values of μ and ν are presented in Table 1.

Based on these experiments, we may state that learning rates for online adaptation are about 20 to 200 times bigger than those corresponding to batch training; learning rates in the exponential layer are between 10^2 and 10^5 times bigger than those of the linear layer.

5.3 Weight Initialization. As stated in section 3, the diagonal initialization of the exponential layer in HFFN corresponds to starting the training from a linear model. To assess possible advantages of such initialization, for all the considered scenarios, we estimated prediction gain r_r as a function of the variance of the weights $\sigma^2_{\mathbf{W}}$ in the exponential layer. All the experiments were performed for both the gaussian and uniform distributions.

The results showed that the initial weight distribution had no influence on the prediction gain (the same results were obtained in Thimm & Fiesler, 1997a, for higher-order perceptrons). For almost all cases (excepted the wind and loma data sets), the prediction gain decreased with an increase in weight variance. To visualize this behavior, we used relative prediction gain r_r expressed by

$$r_r = r_p^R - r_p^D \text{ [dB]}, \tag{5.10}$$

where r_p^R corresponds to prediction gain obtained for random initialization and r_p^D to prediction gain obtained in previous experiments with diagonal initialization. Example curves are presented in Figure 3 for the volterra1 data set.

Since the shapes of these curves are similar for almost all signals, in further experiments only two distinctive initialization schemes were considered: diagonal initialization if the relative prediction gain was smaller than 0 and random initialization when $\sigma^2_{\mathbf{W}}$ was close to 0 (or even $\mathbf{W}^{(1)} = \mathbf{0}$) otherwise.

To perform a quantitative comparison of these results for all the signals, we calculated relative prediction gains related to the random and diagonal

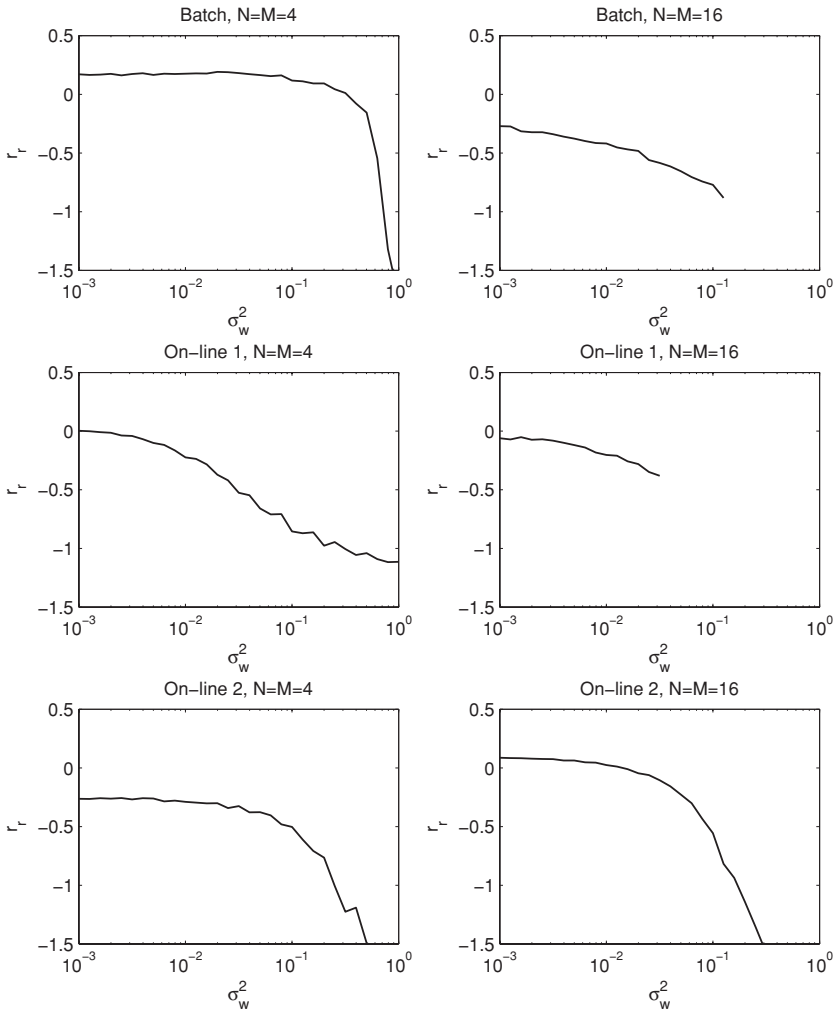

Figure 3: Relative prediction gain r_r as a function of initial weight variance $\sigma^2_{\mathbf{W}}$.

initialization and the corresponding mean values. The results are presented in Table 2, where R(∗) and D(∗) indicate, respectively, the number of random and diagonal initializations for a given scenario and E[r_r] is the corresponding mean relative prediction gain.

Based on these results, we observe that:

- For small-scale networks, there is almost no difference in the performance between the two initialization schemes (the number of diagonal and random initializations is comparable).

Table 2: Mean Relative Prediction Gains for Three Scenarios and All Signals.

	$N = M = 4$		$N = M = 16$	
Batch training	R(5): $E[r_r] = 0.15$	D(7): $E[r_r] = -0.15$	R(3): $E[r_r] = 0.46$	D(9): $E[r_r] = -0.28$
Online 1 adaptation	R(6): $E[r_r] = 0.15$	D(5): $E[r_r] = -1.32$	R(2): $E[r_r] = 0.15$	D(9): $E[r_r] = -1.04$
Online 2 adaptation	R(6): $E[r_r] = 0.62$	D(6): $E[r_r] = -2.97$	R(4): $E[r_r] = 0.61$	D(8): $E[r_r] = -0.89$

- For large-scale networks, diagonal initialization should be preferred (in 9 cases out of 12, diagonal initialization performs better than the random one).
- In general, the prediction gain for the diagonal initialization is higher than the one obtained using random initialization (even if for a particular scenario the number of diagonal and random initializations is comparable).

5.4 HFFN Performances. The performances of a HFFN were next evaluated and compared to those obtained by an FFN with an identical (number of layers and neurons) structure. In all the simulations, the optimal learning rates and initialization parameters found in the two previous sections were used. These optimal settings were also estimated and applied to FFN networks.

As a measure of performance, we used relative prediction gain (on validation set) expressed as

$$r_r = r_p^H - r_p^F \text{ [dB]}, \tag{5.11}$$

where r_p^H is a prediction gain obtained by an HFFN and r_p^F corresponds to that of an FFN network. The results (relative as well as absolute prediction gains) for the three scenarios and two networks (4 and 16 neurons) are shown in, respectively, Tables 3 and 4.[3]

Based on those experiments, we may state:

- **Batch training.** For all the synthetic signals (except lorenz and vdpol), the HFFN performed better than FFN (relative gain up to 5.34 dB for $N = M = 4$ and up to 2.35 dB for $N = M = 16$). For the lorenz and vdpol signals, there was a loss in performance compared to FFN, but since the absolute prediction gain (r_p^H) was in the range of 20 dB, this loss is negligable. For real-life signals, in three ($N = M = 4$) and four ($N = M = 16$) cases out of five, the FFN performed better. This is due to the greater regression-type modeling capabilities of the FFN (based on the universal approximation theorem) compared to the HFFN.

[3]Since there are only 295 samples available in the sunspots data set, there are no corresponding results for the Online 1 scenario (1000 samples are required).

Table 3: Prediction Gains for $N = M = 4$.

$N = M = 4$	r_r			r_p^H		
Signal Type	Batch	Online 1	Online 2	Batch	Online 1	Online 2
ar4	0.19	1.59	1.69	7.50	5.10	4.12
volterra1	0.88	2.49	2.54	2.79	2.58	0.81
volterra2	0.90	4.15	2.42	5.58	13.3	5.08
non1	4.96	0.15	1.40	−1.60	−0.83	0.68
non2	5.34	0.78	1.36	−0.93	−0.01	−0.21
vdpol	−1.53	2.44	1.91	26.4	20.1	15.5
lorenz	−12.0	−8.76	−1.55	23.2	11.9	11.5
dryer	−0.95	0.62	0.05	16.2	12.7	11.9
handel	−0.51	0.07	−0.23	8.35	7.18	6.62
loma	6.11	1.78	0.74	−0.15	10.7	0.31
sunspots	−0.26	—	−3.31	7.90	—	3.44
wind	0.23	0.54	3.45	8.42	8.90	2.20

Table 4: Prediction Gains for $N = M = 16$.

$N = M = 16$	r_r			r_p^H		
Signal Type	Batch	Online 1	Online 2	Batch	Online 1	Online 2
ar4	0.05	1.32	1.71	6.42	3.24	1.37
volterra1	0.06	1.61	1.26	1.75	0.81	−0.37
volterra2	0.37	1.05	0.58	3.67	12.54	1.91
non1	2.35	0.46	0.78	0.29	−0.19	−1.76
non2	0.88	0.62	2.00	0.09	−0.18	−0.13
vdpol	−2.42	−2.28	−0.93	23.9	10.4	9.41
lorenz	−12.5	−6.41	−0.19	19.6	9.25	5.43
dryer	−0.78	−0.18	−1.71	16.1	10.0	2.50
handel	−0.52	0.42	0.88	9.12	7.06	2.59
loma	2.40	1.64	0.82	0.04	8.65	0.22
sunspots	−0.20	—	−1.35	7.09	—	3.34
wind	−0.93	0.37	3.88	7.26	9.46	2.29

These results confirm that for real-life signals, to obtain the best model match, there is a need to use a much more complex model than the one proposed in equation 3.4. In general, for signals where the HFFN performs better, the mean relative gain was 2.65 dB ($N = M = 4$) and 1.02 dB ($N = M = 16$). For other signals (except lorenz and vdpol), the corresponding losses were −0.57 dB ($N = M = 4$) and −0.60 dB ($N = M = 16$).

- **Online adaptation.** For the Online 1 adaptation scheme, the HFFN performed better than FFN in 10 ($N = M = 4$) and 8 ($N = M = 16$) cases out of 11. When the Online 2 scenario was considered, there were 9 ($N = M = 4$) and 8 ($N = M = 16$) cases out of 12 where the

relative gain was positive. To explain this behavior in the case of real-life signals, we may deduce that since the modeling capabilities of an HFFN are derived from different criteria than are those for a FFN, the positive relative gain is due to faster convergence of the HFFN learning algorithm compared to FFN. In other words, for real-time applications where the number of available samples and processing time are constrained, there is a clear indication that an HFFN structure should be preferred.

- **Difficult signals.** For the three data sets non1, non2, and loma, the absolute prediction gains were smaller or close to zero, independent of the learning scenario. In these cases, the HFFN structure exhibits a significant advantage over a sigmoidal FFN: the relative prediction gains were increased up to 6.11 dB for $N = M = 4$ and up to 2.40 dB for $N = M = 16$.

6 Conclusion

We have proposed a new homomorphic feedforward network (HFFN) that, despite its close relation to a standard FFN (structure, signal flow, gradient-based learning), can be seen as a generalized (powers are defined over $\mathbb{R}$) Volterra-like system or as a parallel implementation of a bank of homomorphic filters for multiplicative models.

Gradient-based learning for such an architecture as well as an experimental evaluation of the choice of initial values of the parameters have been addressed. Next, theoretical advantages and disadvantages of the HFFN against an isomorphic sigmoidal FFN have been highlighted. The performances and speed of convergence have been verified through extensive simulations on artificial and real-life signals and compared to those obtained by an FFN. Simulation results have shown a gain in the performance (in terms of a relative prediction gain) and convergence speed (particularly for small data sets) for almost all 12 considered signals and both online adaptation schemes. For batch training, there has been a small loss in performance for real-life signals, leading to the conclusion that more complex modeling capabilities are required.

Extensions of the HFFN include analytic derivation of optimal learning rates, extension to the complex domain, verification of different approaches to obtain a multilayer network, and applications for homomorphic filtering of convolved signals.

References

Billings, S., Jamaluddin, H., & Chen, S. (1992). Properties of neural networks with applications to modelling non-linear dynamical systems. *International Journal of Control*, *55*(1), 193–224.

Cybenko, G. (1989). Approximation by superpositions of sigmoidal function. *Math. Control Signals Systems*, *2*, 303–314.

Doss, R. (1976). Representations of continuous functions of several variables. *American Journal of Mathematics*, *98*(2), 375–383.

Elsken, T. (1990). Smaller nets may perform better: Special transfer functions. *Neural Networks*, *12*, 627–645.

Fahlman, S. (1988). Faster learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, 38–51. Los Altos, CA: Morgan Kaufmann.

Funahashi, K. (1989). On the approximate realisation of continuous mappings by neural networks. *Neural Networks*, *2*, 183–192.

Hanna, A., & Mandic, D. (2003). A data-reusing nonlinear gradient descent algorithm for a class of complex-valued neural adaptive filters. *Neural Processing Letters*, *17*, 85–91.

Haykin, S. (1994). *Neural networks. A comprehensive foundation*. Upper Saddle River, NJ: Prentice Hall.

Haykin, S., & Li, L. (1995). Nonlinear adaptive prediction of nonstationary signals. *IEEE Transactions on Signal Processing*, *43*(2), 526–535.

Hornik, K. (1990). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*, 251–257.

Jacobs, R. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, *1*, 561–573.

Kolmogorov, A. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable. *Doklady Akademi Nauk SSSR*, *114*, 953–957.

Kuribayashi, M., & Tanaka, H. (2005). Fingerprinting protocol for images based on additive homomorphic property. *IEEE Transactions on Image Processing*, *14*(12), 2129–2139.

Lindquist, C., & Mukherjee, S. (1994). A homomorphic approach to companding. In *Proceedings of the 28th Asilomar Conference on Signals, Systems and Computers* (vol. 2, pp. 933–937). Pacific Grove, CA.

Lippman, R. (1987). An introduction of computing with neural nets. *IEEE Acoust. Speech Signal Processing Magazine*, *4*(2), 4–22.

Lorenz, E. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, *20*(2), 130–141.

Mandic, D. (2004). A generalized normalized gradient descent Algorithm. *IEEE Signal Processing Letters*, *11*(2), 115–118.

Mandic, D., & Chambers, J. (2000a). Relationship between the a priori and a posteriori errors in nonlinear adaptive neural filters. *Neural Computation*, *12*, 1285–1292.

Mandic, D., & Chambers, J. (2000b). Towards the optimal learning rate for backpropagation. *Neural Processing Letters*, *11*, 1–5.

Mandic, D., & Chambers, J. (2001). *Recurrent Neural Networks for Prediction*. New York: Wiley.

Mandic, D., Hanna, A., & Razaz, M. (2001). A normalized gradient descent algorithm for nonlinear adaptive filters using a gradient adaptive step size. *IEEE Signal Processing Letters*, *8*(11), 295–297.

Marenco, A., & Madisetti, V. (1996). A constructive deconvolution procedure of bandpass signals by homomorphic analysis. In *Proceedings of the Geoscience and Remote Sensing Symposium, IGARSS'96* (Vol. 3, pp. 1592–1596). Lincoln, NE.

Mathews, V. (1991). Adaptive polynomial filters. *IEEE Signal Processing Magazine*, *8*(3), 10–26.

Mathews, V., & Xie, Z. (1993). A stochastic gradient adaptive filter with gradient adaptive step size. *IEEE Transactions on Signal Processing*, *41*(6), 2075–2087.

Narendra, K., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–26.

Oppenheim, A. (1965, March 31). *Superposition in a class of non-linear systems* (Research Report No. 432). Cambridge, Mass.: Research Laboratory of Electronics, MIT.

Oppenheim, A. (1969). Speech analysis-synthesis system based on homomorphic filtering. *Journal of the Acoustical Society of America*, *45*(2), 458–465.

Oppenheim, A., Kopec, G., & Tribolet, J. (1976). Signal analysis by homomorphic prediction. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, *ASSP-24*(4), 327–332.

Oppenheim, A., & Schafer, R. (1968). Homomorphic analysis of speech. *IEEE Transactions on Audio Electroacoustics*, *AU-16*, 221–226.

Oppenheim, A., & Schafer, R. (1975). *Digital signal processing*. Upper Saddle River, NJ: Prentice Hall.

Oppenheim, A., Schafer, R., & Stockham, T. (1968). Nonlinear filtering of multiplied and convolved signals. *Proceedings of the IEEE*, *56*(8), 1264–1291.

Rumelhart, D., Hinton, G., & Williams, R. (1986). *Parallel distributed processing, Vol. 1: Foundations*. Cambridge, MA: MIT Press.

Ryu, H., Lee, K., & Kwon, B. (2004). Change detection for urban analysis with high-resolution imagery: Homomorphic filtering and morphological operation approach. In *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium, IGARSS'04* (Vol. 4, pp. 2662–2664). Piscataway, NJ: IEEE Press.

Sabry-Rizk, M., Zgallai, W., Hardiman, P., & O'Riordan, J. (1995). Blind deconvolution homomorphic analysis of abnormalities in ECG signals. In *IEE Colloquium on Blind Deconvolution—Algorithms and Applications*. London.

Schafer, R., & Rabiner, L. (1970). System for automatic analysis of voiced speech. *Journal of the Acoustical Society of America*, *47*(2), 634–648.

Söderström, T., & Stoica, P. (1994). *System identification*. Upper Saddle River, NJ: Prentice Hall.

Stoffa, P., Buhl, P., & Bryan, G. (1974a). The application of homomorphic deconvolution to shallow-water marine seismology—Part I: Models. *Geophysics*, *39*, 401–416.

Stoffa, P., Buhl, P., & Bryan, G. (1974b). The application of homomorphic deconvolution to shallow-water marine seismology—Part II: Real Data. *Geophysics*, *39*, 417–426.

Takens, F. (1981). On the numerical determination of the dimension of an attractor. In B. L. J. Braaksma, H. W. Broer, & F. Takers (Eds.), *Dynamical systems and turbulence.* Berlin: Springer.

Thimm, G., & Fiesler, E. (1997a). High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, *8*(2), 349–359.

Thimm, G., & Fiesler, E. (1997b, April). *Optimal settings of weights, learning rate, and gain* (Research Report). Martigny, Switzerland: Dalle Molle Institute for Perspective Artificial Intelligence.

Thimm, G., Moerland, P., & Fiesler, E. (1996). The interchangeability of learning rate and gain in backpropagation neural networks. *Neural Computation*, *8*(2), 451–460.

Ulrych, T. (1971). Application of homomorphic deconvolution to seismology. *Geophisics*, *36*, 650–660.

Van Ooyen, A., & Nienhuis, B. (1992). Improving the convergence of the back-propagation algorithm. *Neural Networks*, *5*(3), 465–472.

Wiggins, S. (1990). *Introduction to applied nonlinear dynamical systems and chaos*. New York: Springer-Verlag.

Received December 15, 2006; accepted May 27, 2007.