January 2002

# Neuron-adaptive higher order neural-network models for automated financial data modeling

M. Zhang
*Christoper Newport University, Virginia*

S. Xu
*University of Tasmania*

J. Fulcher
*University of Wollongong*, john@uow.edu.au

# Neuron-adaptive higher order neural-network models for automated financial data modeling


## Abstract

Real-world financial data is often nonlinear, comprises high-frequency multipolynomial components, and is discontinuous (piecewise continuous). Not surprisingly, it is hard to model such data. Classical neural networks are unable to automatically determine the optimum model and appropriate order for financial data approximation. We address this problem by developing neuron-adaptive higher order neural-network (NAHONN) models. After introducing one-dimensional (1-D), two-dimensional (2-D), and n-dimensional NAHONN models, we present an appropriate learning algorithm. Network convergence and the universal approximation capability of NAHONNs are also established. NAHONN Group models (NAHONGs) are also introduced. Both NAHONNs and NAHONGs are shown to be "open box" and as such are more acceptable to financial experts than classical (closed box) neural networks. These models are further shown to be capable of automatically finding not only the optimum model, but also the appropriate order for specific financial data.



## Disciplines

Physical Sciences and Mathematics

## Publication Details

This paper originally appeared as: Zhang, M, Xu, S and Fulcher, J, Neuron-adaptive higher order neural-network models for automated financial data modeling, IEEE Transactions on Neural Networks, January 2002, 13(1)1, 188-204.

# Neuron-Adaptive Higher Order Neural-Network Models for Automated Financial Data Modeling

Ming Zhang, *Senior Member, IEEE*, Shuxiang Xu, and John Fulcher, *Senior Member, IEEE*

***Abstract*—Real-world financial data is often nonlinear, comprises high-frequency multipolynomial components, and is discontinuous (piecewise continuous). Not surprisingly, it is hard to model such data. Classical neural networks are unable to automatically determine the optimum model and appropriate order for financial data approximation. We address this problem by developing neuron-adaptive higher order neural-network (NAHONN) models. After introducing one-dimensional (1-D), two-dimensional (2-D), and $n$-dimensional NAHONN models, we present an appropriate learning algorithm. Network convergence and the universal approximation capability of NAHONNs are also established. NAHONN *Group* models (NAHONGs) are also introduced. Both NAHONNs and NAHONGs are shown to be "open box" and as such are more acceptable to financial experts than classical (closed box) neural networks. These models are further shown to be capable of automatically finding not only the optimum model, but also the appropriate order for specific financial data.**

***Index Terms*—Financial modeling, higher order neural networks, neural-network groups, piecewise functions, polynomials.**

## I. Introduction

MODELING and predicting financial data using traditional statistical approaches has only been partially successful [4], [20]. Accordingly, researchers have turned to alternative approaches in recent times, most notably artificial neural networks (ANNs) [1]. The last few years have seen the rise of specialist conferences, special journal issues (and indeed journals), and books in intelligent financial modeling (e.g., [2], [23], [25]).

Standard ANN models cannot deal with discontinuities $\{f(x) \neq \lim_{\Delta x \to 0} f(x + \Delta x)\}$ in the input training data. Furthermore, they suffer from the following limitations [4], [5].

- They do not always perform well because of the complexity (higher frequency components and higher order nonlinearity) of the economic data being simulated, and
- The neural networks function as "black boxes," and are thus unable to provide explanations for their behavior (although some recent successes have been reported with rule extraction from trained ANNs [10], [18]).

This latter feature is viewed as a disadvantage by users, who would rather be given a rationale for the simulation at hand.

In an effort to overcome the limitations of conventional ANNs, some researchers have turned their attention to higher order neural network (HONN) models [14], [16], [21], [32]. HONN models are able to provide some rationale for the simulations they produce, and thus can be regarded as "open box" rather than "black box." Moreover, HONNs are able to simulate higher frequency, higher order nonlinear data, and consequently provide superior simulations compared to those produced by ANNs. Polynomials or linear combinations of trigonometric functions are often used in the modeling of financial data. Using HONN models for financial simulation and/or modeling would lead to open box solutions, and hence be more readily accepted by target users (i.e., financial experts).

This was the motivation for developing the *Polynomial* HONNs (PHONNs) for economic data simulation [27]. This idea has been subsequently extended into first *Group* PHONN (PHONNG) models for financial data simulation [28], and second *trigonometric PHONNG* models for financial prediction [25], [26], [32].

Real-world financial data often comprises high-frequency multipolynomial components, and is discontinuous (or piecewise continuous). Not surprisingly, it is difficult to automatically determine the best model for analyzing such financial data. If a suitable model can be found, it is still very hard to determine the best order for financial data simulation. Conventional neural-network models are unable to find the optimal model (and order) for financial data approximation. In order to solve this problem, autoselection financial modeling software has been developed [21], likewise adaptive higher order neural networks have also been studied [22], [23]. Neuron-adaptive feedforward neural-network groups and neuron-adaptive higher order neural-network groups have also been applied to this problem [30], [31]. So far however, the results have been limited. Hence one motivation for the present study was to automatically select the optimum higher order neural-network model (and order) appropriate for analyzing financial data.

Many authors have addressed the problem of approximation by feedforward neural networks (FNNs) (e.g., [3], [11]–[13], [17], [19], [20]). FNNs have been shown to be capable of approximating generic classes of functions. For example, Hornik established that FNNs with a single hidden layer can uniformly approximate continuous functions on compact subsets, provided that the activation function is locally Riemann integrable and

Manuscript received March 1, 2001; revised August 20, 2001. This work was supported by Fujitsu Research Laboratories, Japan, and by grants from the University of Western Sydney and the Australian Research Council, as well as from the Institute for Mathematical Modeling and Computational Systems, University of Wollongong, Australia. The theoretical work contained herein was performed while M. Zhang was the recipient of a USA National Research Council Senior Research Associateship with the National Environmental Satellite Data and Information Service of NOAA.

M. Zhang is with the Department of Physics, Computer Science, and Engineering, Christopher Newport University, Newport News, VA 23606 USA (e-mail: mzhang@pcs.cnu.edu).

S. Xu is with the School of Computing, University of Tasmania, Launceston, TAS 7250, Australia (e-mail: Shuxiang.Xu@utas.edu.au).

J. Fulcher is with the School of Information Technology and Computer Science, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: john@cs.uow.edu.au).

Publisher Item Identifier S 1045-9227(02)00361-2.

nonpolynomial [13]. Park and Sandberg showed that radial basis function (RBF) networks are broad enough for universal approximation [20]. Leshno proved that FNNs can approximate any continuous function to any degree of accuracy if and only if the network's activation function is nonpolynomial [17]. Last, Chen explored the universal approximation capability of FNNs to continuous functions, functionals and operators [8], [9].

Two problems remain, however.

- The activation functions employed in these studies are sigmoid, generalized sigmoid, RBF, and the like. One characteristic of such activation functions is that they are all fixed (with no free parameters), and thus cannot be adjusted to adapt to *different* approximation problems.
- In real-world applications such as financial data simulation, the function to be approximated can be a piecewise continuous one. A *continuous* function approximation is unable to deal with nonlinear and discontinuous data, such as that commonly encountered in economic data simulation.

To date there have been few studies which emphasize the setting of free parameters in the activation function. Networks which employ adaptive activation functions seem to provide better fitting properties than classical architectures which use fixed activation function neurons. Vecci studied the properties of an FNN which is able to adapt its activation function by varying the control points of a Catmull–Rom cubic spline [24]. Their simulations confirm that this specialized learning mechanism allows the effective use of the network's free parameters. Chen and Chang adjust the real variables $a$ (gain) and $b$ (slope) in the generalized sigmoid activation function during learning [7]. Compared with classical FNNs in modeling static and dynamical systems, they showed that an *adaptive* sigmoid leads to improved data modeling. Campolucci showed that a neuron-adaptive activation function built using a piecewise approximation with suitable cubic splines can have arbitrary shape, and furthermore leads to reduced neural-network size [6]. In other words, connection complexity is traded off against activation function complexity. Other authors have also studied the properties of neural networks with neuron-adaptive activation functions [15], [24]. The development of a new neural-network model which utilizes an adaptive neuron activation function, and then uses this new model to automatically determine both the best model and appropriate order for different financial data is the second motivation of this paper.

While Hornik and Leshno proved the approximation ability of FNNs for any continuous function, Zhang established neural-network group models to approximate piecewise continuous functions (as a means of simulating discontinuous financial data) [28]. A neural-network group is a generalized neural-network set in which each element is a neural network, and for which product and addition of any two elements have been defined. Herein lies a problem though: if the piecewise continuous function to be approximated comprises an *infinite* number of sections of continuous functions, then the neural-network group will need to contain an *infinite* number of neural networks! This makes the simulation very complicated, because each continuous section would have to be approximated independently and separately. Moreover, no learning algorithm yet exists for neural-network groups. It is therefore necessary to consider approximating a piecewise continuous function with a *single* neural network (rather than a group). The third motivation for this paper is thus to develop a methodology for automatically finding the optimal model (and appropriate order) for simulating high-frequency multipolynomial, discontinuous (or piecewise continuous) financial data.

Section I of this paper introduced the background to and motivations for this research. In Section II, neuron-adaptive higher order neural network (NAHONN) models with neuron-adaptive activation functions are developed, in order to approximate any continuous (or piecewise) function, to any degree of accuracy. Our work differs from previous studies in the following ways.

- An adaptive activation function learning algorithm is developed in Section III, which is capable of automatically determining both the optimal model and correct order for financial data modeling, and
- In SectionIV, we present a theoretical proof which shows that a single NAHONN is able to approximate any piecewise continuous function, thereby guaranteeing better modeling results and less error.

In order to simulate some specialized financial functions, we develop NAHONN group models (NAHONGs) in Section V. In Section VI, we present specialized NAHONN models, namely trigonometric (T-NAHONN), polynomial and trigonometric (PT-NAHONN), trigonometric exponent and sigmoid (TES-NAHONN), as well as their group counterparts. These specialized NAHONN models are appropriate for financial data simulation involving higher frequency, higher order nonlinear, multipolynomial, discontinuous, nonsmooth, and other such features. Experimental results obtained using NAHONNs and NAHONGs are presented in Section VII, and finally in Section VIII we present brief conclusions.

## II. NAHONN

### A. One-Dimensional (1-D) NAHONN Definition

Let

| | |
|---|---|
| $i$ | $i$th neuron in layer-$k$; |
| $k$ | $k$th layer of the neural network ($k$ will be used later in the learning algorithm proof); |
| $h$ | $h$th term in the Neural-network Activation Function (NAF); |
| $s$ | maximum number of terms in the NAF; |
| $x$ | first neural-network input; |
| $y$ | second neural-network input; |
| $\text{net}_{i,k}$ | input or internal state of the $i$th neuron in the $k$th layer; |
| $w_{i,j,k}$ | weight that connects the $j$th neuron in layer $k-1$ with the $i$th neuron in layer $k$ $j$ *will be used* in *the two*—dim*ensional NAHONN formula*; |
| $o_{i,k}$ | value of the output from the $i$th neuron in layer–$k$. |

The 1-D NAF is defined as

$$\text{NAF: } \Psi_{i,k}(\text{net}_{i,k}) = o_{i,k}(\text{net}_{i,k}) = \sum_{h=1}^{s} f_{i,k,h}(\text{net}_{i,k}). \tag{2.1}$$

Suppose

$$\begin{aligned} s &= 4 \\ f_{i,k,1}(\text{net}_{i,k}) &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,k})) \\ f_{i,k,2}(\text{net}_{i,k}) &= a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,k})} \\ f_{i,k,3}(\text{net}_{i,k}) &= a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (\text{net}_{i,k})}} \\ f_{i,k,4}(\text{net}_{i,k}) &= a4_{i,k} \cdot (\text{net}_{i,k})^{b4_{i,k}}. \end{aligned} \tag{2.2}$$

The 1-D NAF then becomes

$$\begin{aligned} &\Psi_{i,k}(\text{net}_{i,k}) \\ &= \sum_{h=1}^{4} f_{i,k,h}(\text{net}_{i,k}) \\ &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,k})) + a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,k})} \\ &\quad + a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (\text{net}_{i,k})}} + a4_{i,k} \cdot (\text{net}_{i,k})^{b4_{i,k}} \end{aligned} \tag{2.3}$$

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, a2_{i,k}, b2_{i,k}, a3_{i,k}, b3_{i,k}, a4_{i,k}, b4_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

Let

$$\text{net}_{i,k} = w_{i,x,k} \cdot x.$$

The 1-D NAHONN is defined as

$$\begin{aligned} &\text{NAHONN (1-D):} \\ &\sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(w_{i,x,k} \cdot x) \\ &\quad = \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{s} f_{i,k,h}(w_{i,x,k} \cdot x). \end{aligned} \tag{2.4}$$

Let

$$\begin{aligned} s &= 4 \\ f_{i,k,1}(\text{net}_{i,k}) &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,k})) \\ f_{i,k,2}(\text{net}_{i,k}) &= a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,k})} \\ f_{i,k,3}(\text{net}_{i,k}) &= a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (\text{net}_{i,k})}} \\ f_{i,k,4}(\text{net}_{i,k}) &= a4_{i,k} \cdot (\text{net}_{i,k})^{b4_{i,k}} \end{aligned} \tag{2.5}$$

$$\begin{aligned} &\text{NAHONN (1-D):} \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(\text{net}_{i,k}) \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{4} f_{i,k,h}(w_{i,x,k} \cdot x) \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot (a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (w_{i,x,k} \cdot x)) \\ &\quad + a2_{i,k} \cdot e^{-b2_{i,k} \cdot (w_{i,x,k} \cdot x)} \\ &\quad + a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (w_{i,x,k} \cdot x)}} \\ &\quad + a4_{i,k} \cdot (\text{net}_{i,k})^{b4_{i,k}}) \end{aligned} \tag{2.6}$$

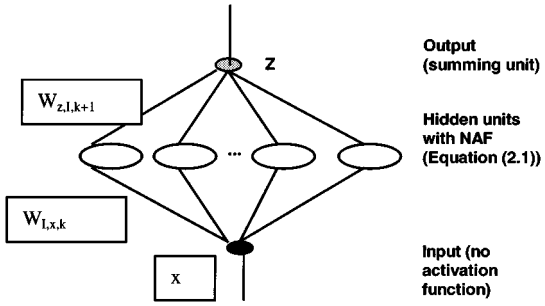


Fig. 1. 1-D NAHONN structure.

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, a2_{i,k}, b2_{i,k}, a3_{i,k}, b3_{i,k}, a4_{i,k}, b4_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

The network structure of a 1-D NAHONN is the same as that of a multilayer FNN. That is, it consists of an input layer with *one* input-unit, an output layer with one output-unit, and one hidden layer consisting of intermediate processing units. A typical 1-D NAHONN architecture is depicted in Fig. 1. Now while there is no activation function in the input layer and the output neuron is a summing unit only (linear activation), the activation function in the hidden units is the 1-D neuron-adaptive higher order neural-network NAF defined by (2.1). The 1-D NANONN is described by (2.4).

### *B. Two-Dimensional (2-D) NAHONN Definition*

Let

$\text{net}_{i,x,k} =$ input of the $x$ neuron in the $k$th layer;

$\text{net}_{i,y,k} =$ input of the $y$ neuron in the $k$th layer.

The 2-D NAF is defined as

$$\begin{aligned} &\text{NAF: } \Psi_{i,k}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) \\ &\quad = O_{i,k}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) \\ &\quad = \sum_{h=1}^{s} f_{i,k,h}(\text{net}_{i,x,k}, \text{net}_{i,y,k}). \end{aligned} \tag{2.7}$$

Suppose

$$\begin{aligned} s &= 4 \\ f_{i,k,1}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,x,k})) \\ &\quad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (\text{net}_{i,y,k})) \\ f_{i,k,2}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,x,k})} \\ &\quad \cdot e^{-d2_{i,k} \cdot (\text{net}_{i,x,k})} \\ f_{i,k,3}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (\text{net}_{i,x,k})}} \\ &\quad \cdot \frac{1}{1 + e^{-d3_{i,k} \cdot (\text{net}_{i,y,k})}} \\ f_{i,k,4}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a4_{i,k} \cdot (\text{net}_{i,x,k})^{b4_{i,k}} \\ &\quad \cdot (\text{net}_{i,y,k})^{d4_{i,k}}. \end{aligned} \tag{2.8}$$

The 2-D NAHONN then becomes

$$\begin{aligned}\Psi_{i,k}(\text{net}_{i,k}) &= \sum_{h=1}^{4} f_{i,k,h}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) \\ &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,x,k})) \\ &\quad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (\text{net}_{i,y,k})) \\ &\quad + a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,x,k})} \cdot e^{-d2_{i,k} \cdot (\text{net}_{i,y,k})} \\ &\quad + a3_{i,k} \cdot \frac{1}{1+e^{-b3_{i,k} \cdot (\text{net}_{i,x,k})}} \cdot \frac{1}{1+e^{-d3_{i,k} \cdot (\text{net}_{i,y,k})}} \\ &\quad + a4_{i,k} \cdot (\text{net}_{i,x,k})^{b4_{i,k}} \cdot (\text{net}_{i,y,k})^{d4_{i,k}} \end{aligned} \tag{2.9}$$

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, d1_{i,k}, e1_{i,k}, a2_{i,k}, b2_{i,k}, d2_{i,k} \\ a3_{i,k}, b3_{i,k}, d3_{i,k}, a4_{i,k}, b4_{i,k}, d4_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

Let

$$\text{net}_{i,x,k} = w_{i,x,k} \cdot x$$
$$\text{net}_{i,y,k} = w_{i,y,k} \cdot y.$$

The 2-D NAHONN is defined as

NAHONN (2-D):

$$\sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) = \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{s} f_{i,k,h}(w_{i,x,k} \cdot x, w_{i,y,k} \cdot y). \tag{2.10}$$

Let

$$\begin{aligned} s &= 3 \\ f_{i,k,1}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\text{net}_{i,x,k})) \\ &\quad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (\text{net}_{i,y,k})) \\ f_{i,k,2}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a2_{i,k} \cdot e^{-b2_{i,k} \cdot (\text{net}_{i,x,k})} \\ &\quad \cdot e^{-d2_{i,k} \cdot (\text{net}_{i,x,k})} \\ f_{i,k,3}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) &= a3_{i,k} \cdot \frac{1}{1+e^{-b3_{i,k} \cdot (\text{net}_{i,x,k})}} \\ &\quad \cdot \frac{1}{1+e^{-d3_{i,k} \cdot (\text{net}_{i,y,k})}} \end{aligned} \tag{2.11}$$

NAHONN (2-D)

$$\begin{aligned} &\sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(\text{net}_{i,x,k}, \text{net}_{i,y,k}) \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{3} f_{i,k,h}(w_{i,x,k} \cdot x, w_{i,y,k} \cdot y) \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot \Bigg( \Big( a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (w_{i,x,k} \cdot x)) \\ &\quad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (w_{i,y,k} \cdot y)) + a2_{i,k} \cdot e^{-b2_{i,k} \cdot (w_{i,x,k} \cdot x)} \\ &\quad \cdot e^{-d2_{i,k} \cdot (w_{i,y,k} \cdot y)} + a3_{i,k} \\ &\quad \cdot \frac{1}{1+e^{-b3_{i,k} \cdot (w_{i,x,k} \cdot x)}} \cdot \frac{1}{1+e^{-d3_{i,k} \cdot (w_{i,y,k} \cdot y)}} \Bigg) \Bigg) \end{aligned} \tag{2.12}$$

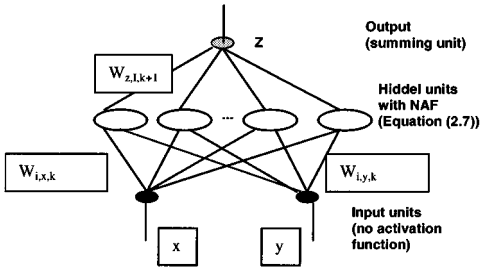


Fig. 2. 2-D NAHONN structure.

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, d1_{i,k}, e1_{i,k}, a2_{i,k}, b2_{i,k}, d2_{i,k} \\ a3_{i,k}, b3_{i,k}, d3_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

The network structure of the 2-D NAHONN is the same as that of a multilayer FNN. That is, it consists of an input layer with *two* input-units, an output layer with one output-unit, and one hidden layer consisting of intermediate processing units. A typical 2-D NAHONN architecture is depicted in Fig. 2. Again, while there is no activation function in the input layer and the output neuron is a summing unit (linear activation), the activation function for the hidden units is the 2-D NAHONN defined by (2.7). The 2-D NANONN is described by (2.10).

### *C. $m$-Dimensional NAHONN Definition*

The $m$-dimensional NAF is

$$\Psi_{i,k}(\text{net}_{i,k}) = \sum_{h=1}^{s} f_{i,k,h}(\text{net}_{i,1,k}, \text{net}_{i,2,k}, \ldots, \text{net}_{i,m,k}). \tag{2.13}$$

Let

$$\begin{aligned} \text{net}_{i,1,k} &= w_{i,1,k} \cdot x_1 \\ \text{net}_{i,2,k} &= w_{i,2,k} \cdot x_2 \\ &\cdots \\ \text{net}_{i,n,k} &= w_{i,m,k} \cdot x_m. \end{aligned}$$

The $m$-dimensional NAHONN is defined as

NAHONN ($m$-Dimensional):

$$\begin{aligned} &\sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(\text{net}_{i,1,k}, \text{net}_{i,2,k}, \ldots, \text{net}_{i,m,k}) \\ &= \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{s} f_{i,k,h} \\ &\quad \times (w_{i,1,k} \cdot x_1, w_{i,2,k} \cdot x_2 \ldots, w_{i,m,k} \cdot x_m). \end{aligned} \tag{2.14}$$

The network structure of an $m$-dimensional NAHONN is the same as that of a multilayer FNN. That is, it consists of an input layer with $m$ input-units, an output layer with one output-unit, and one hidden layer consisting of intermediate processing units. A typical 2-D NAHONN architecture is depicted in Fig. 2. Again, while there is no activation function in the input layer and the output neuron is a summing unit (linear activation), the activation function for the hidden units is the $m$-dimensional neuron-adaptive higher order neural network NAF defined by (2.13). The $m$-dimensional NANONN Fig. 3 is described by (2.14).

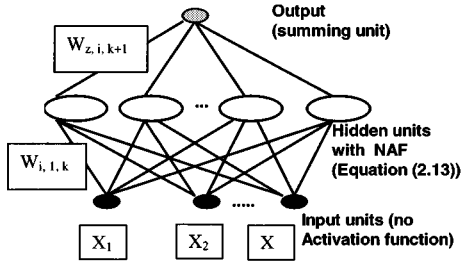


Fig. 3. $m$-dimensional NAHONN structure.

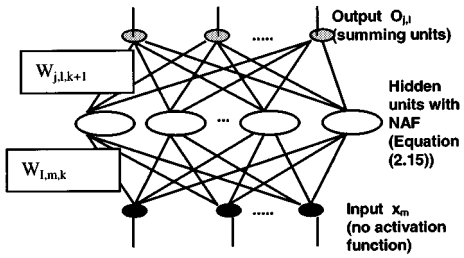


Fig. 4. Multi $m$-dimensional NAHONN structure.

### D. Multi $m$-Dimensional NAHONN Definition

The *multi* $m$-dimensional NAHONN is defined as follows:

$$O_{j,l} = \sum_{i=0}^{n} w_{j,i,k+1} \cdot \subset \Psi_{i,k}(\mathrm{net}_{i,1,k}, \mathrm{net}_{i,2,k}, \ldots, \mathrm{net}_{i,m,k}) = \sum_{i=0}^{n} w_{j,i,k+1} \cdot \sum_{h=1}^{s} f_{i,k,h}(w_{i,1,k} \cdot x_1, w_{i,2,k} \cdot x_2 \ldots, w_{i,m,k} \cdot x_m) \tag{2.15}$$

where

$l$: output layer;
$j$: $j$th output in the output layer;
$O_{j,l}$: one output of the multi $m$-dimensional NAHONN.

The structure of the multi $m$-dimensional NAHONN is shown in Fig.4.

## III. NAHONN Learning Algorithm

Our learning algorithm is based on the steepest descent gradient rule [22]. However, as the variables in the hidden layer activation function are able to be adjusted, NAHONN provides more flexibility and more accurate approximation than traditional higher order (and indeed NAHONN includes traditional higher order FNN (fixed activation function) FNNs as a special case).

We use the following notations:

| | |
|---|---|
| $\mathrm{net}_{i,k}$ | input or internal state of the $i$th neuron in the $k$th layer; |
| $w_{i,j,k}$ | weight that connects the $j$th neuron in layer $k-1$ with the $i$th neuron in layer $k$; |
| $o_{i,k}$ | value of the output from the $i$th neuron in layer $k$; |
| $a1, b1, c1, a2, b2, a3, b3, a4, b4$ | adjustable variables in the activation function; |
| $\theta_{i,k}$ | threshold value of the $i$th neuron in the $k$th layer; |
| $d_j$ | $j$th desired output value; |
| $r$ | iteration number; |
| $\eta$ | learning rate; |
| $m$ | total number of output layer neurons; |
| $l$ | total number of network layers. |

The 1-D NAF is defined as

$$\text{NAF: } \Psi_{i,k}(\mathrm{net}_{i,k}) = O_{i,k}(\mathrm{net}_{i,k}) = \sum_{h=1}^{s} f_{i,k,h}(\mathrm{net}_{i,k}). \tag{3.1}$$

First of all, the input–output relation of the $i$th neuron in the $k$th layer can be described by

$$\mathrm{net}_{i,k} = \sum_{j} [w_{i,j,k} o_{j,k-1}] - \theta_{i,k} \tag{3.2}$$

where $j$ is the number of neurons in layer $k-1$, and

$$o_{i,k} = \Psi(\mathrm{net}_{i,k}) = a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot \mathrm{net}_{i,k}) + a2_{i,k} \cdot e^{-b2_{i,k} \cdot \mathrm{net}_{i,k}} + \frac{a3_{i,k}}{1 + e^{-b3_{i,k} \cdot \mathrm{net}_{i,k}}} + a4_{i,k} \cdot (\mathrm{net}_{i,k})^{b4_{i,k}}. \tag{3.3}$$

To train the NAHONN, the following energy function is adopted:

$$E = \frac{1}{2} \sum_{j=1}^{m} (d_j - o_{j,l})^2. \tag{3.4}$$

$E$ is the sum of the squared errors between the actual network output and the desired output for all input patterns. In (3.4), $m$ is the total number of output layer neurons, and $l$ is the total number of constructed network layers (here $l = 3$). The aim of learning is to minimize this energy function by adjusting both the weights associated with various interconnections, as well as the variables in the activation function. This can be achieved by using the steepest descent gradient rule expressed as follows [22]:

$$w_{i,j,k}^{(r)} = w_{i,j,k}^{(r-1)} + \eta \frac{\partial E}{\partial w_{i,j,k}} \tag{3.5}$$

$$\theta_{i,k}^{(r)} = \theta_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial \theta_{i,k}} \tag{3.6}$$

$$a1_{i,k}^{(r)} = a1_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial a1_{i,k}} \tag{3.7}$$

$$b1_{i,k}^{(r)} = b1_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial b1_{i,k}} \tag{3.8}$$

$$c1_{i,k}^{(r)} = c1_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial c1_{i,k}} \tag{3.9}$$

$$a2_{i,k}^{(r)} = a2_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial a2_{i,k}} \tag{3.10}$$

$$b2_{i,k}^{(r)} = b2_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial b2_{i,k}} \tag{3.11}$$

$$a3_{i,k}^{(r)} = a3_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial a3_{i,k}} \tag{3.12}$$

$$b3_{i,k}^{(r)} = b3_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial b3_{i,k}} \tag{3.13}$$

$$a4_{i,k}^{(r)} = a4_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial a4_{i,k}} \tag{3.14}$$

$$b4_{i,k}^{(r)} = b4_{i,k}^{(r-1)} + \eta \frac{\partial E}{\partial b4_{i,k}}. \tag{3.15}$$

To derive the gradient of $E$ with respect to each adjustable parameter in (3.5)–(3.15), we define

$$\frac{\partial E}{\partial \text{net}_{i,k}} = \delta_{i,k} \tag{3.16}$$

$$\frac{\partial E}{\partial o_{i,k}} = \xi_{i,k}. \tag{3.17}$$

Now, from (3.2), (3.3), (3.16) and (3.17), we have the partial derivatives of $E$ with respect to the adjustable parameters as follows:

$$\frac{\partial E}{\partial w_{i,j,k}} = \frac{\partial E}{\partial \text{net}_{i,k}} \frac{\partial \text{net}}{\partial w_{i,j,k}} = \delta_{i,k} o_{j,k-1} \tag{3.18}$$

$$\frac{\partial E}{\partial \theta_{i,k}} = \frac{\partial E}{\partial \text{net}_{i,k}} \frac{\partial \text{net}}{\partial \theta_{i,k}} = -\delta_{i,k} \tag{3.19}$$

$$\frac{\partial E}{\partial a1_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial a1_{i,k}} = \xi_{i,k} \cdot \sin(b1_{i,k} \cdot \text{net}_{i,k}) \tag{3.20}$$

$$\frac{\partial E}{\partial b1_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial b1_{i,k}} = \xi_{i,k} \cdot a1_{i,k} \cdot \text{net}_{i,k} \cdot \cos(b1_{i,k} \cdot \text{net}_{i,k}) \tag{3.21}$$

$$\frac{\partial E}{\partial c1_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial c1_{i,k}} = \xi_{i,k} \cdot a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot \text{net}_{i,k}) \cdot \ln[\sin(b1_{i,k} \cdot \text{net}_{i,k})] \tag{3.22}$$

$$\frac{\partial E}{\partial a2_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial a2_{i,k}} = \xi_{i,k} e^{-b2_{i,k} \cdot \text{net}_{i,k}^2} \tag{3.23}$$

$$\frac{\partial E}{\partial b2_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial b2_{i,k}} = -\xi_{i,k} \cdot a2_{i,k} \cdot \text{net}_{i,k} \cdot e^{-b2_{i,k} \cdot \text{net}_{i,k}^2} \tag{3.24}$$

$$\frac{\partial E}{\partial a3_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial a3_{i,k}} = \xi_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot \text{net}_{i,k}}} \tag{3.25}$$

$$\frac{\partial E}{\partial b3_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial b3_{i,k}} = \xi_{i,k} \cdot \frac{a3_{i,k} \cdot \text{net}_{i,k} \cdot e^{-b3_{i,k} \cdot \text{net}_{i,k}}}{(1 + e^{-b3_{i,k} \cdot \text{net}_{i,k}})^2} \tag{3.26}$$

$$\frac{\partial E}{\partial a4_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial a4_{i,k}} = \xi_{i,k} \cdot (\text{net}_{i,k})^{b4_{i,k}} \tag{3.27}$$

$$\frac{\partial E}{\partial b4_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial b4_{i,k}} = \xi_{i,k} \cdot a4_{i,k} (\text{net}_{i,k})^{b4_{i,k}} \ln(\text{net}_{i,k}) \tag{3.28}$$

and for (3.16) and (3.17) the following equations can be computed:

$$\delta_{i,k} = \frac{\partial E}{\partial \text{net}_{i,k}} = \frac{\partial E}{\partial o_{i,k}} \frac{\partial o_{i,k}}{\partial \text{net}_{i,k}} = \xi_{i,k} \cdot \frac{\partial o_{i,k}}{\partial \text{net}_{i,k}} \tag{3.29}$$

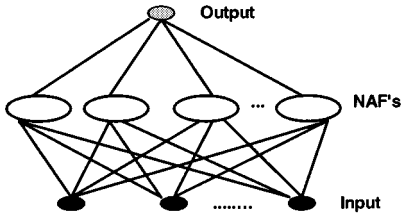


Fig. 5. NANONN with one output.

while

$$\frac{\partial o_{i,k}}{\partial \text{net}_{i,k}} = a1_{i,k} \cdot b1_{i,k} \cdot c1_{i,k} \cdot \sin^{c1_{i,k}-1}(b1_{i,k} \cdot \text{net}_{i,k}) \times \cos(b1_{i,k} \cdot \text{net}_{i,k}) - a2_{i,k} \cdot b2_{i,k} \cdot e^{-b2_{i,k} \cdot \text{net}_{i,k}^2} + \frac{a3_{i,k} \cdot b3_{i,k} \cdot e^{-b3_{i,k} \cdot \text{net}_{i,k}}}{(1 + e^{-b3_{i,k} \cdot \text{net}_{i,k}})^2} + a4_{i,k} b4_{i,k} (\text{net}_{i,k})^{b4_{i,k}-1} \tag{3.30}$$

and

$$\xi_{i,k} = \begin{cases} \sum_j \delta_{j,k+1} w_{j,i,k+1}, & \text{if } 1 \le k < l; \\ o_{i,l} - d_i, & \text{if } k = l. \end{cases} \tag{3.31}$$

We summarize the procedure for performing the above algorithm as follows.

00: Determine the number of hidden units for the network.

10: Initialize all weights $w_{i,j,k}$, threshold values $\theta_{i,k}$ and parameters $a1, b1, c1, a2, b2, a3, b3, a4, b4$ of the hidden units' activation functions.

20: Input a learning example from the training data and calculate the actual outputs of all neurons using the present parameter values, according to (3.2) and (3.3).

30: Evaluate $\delta_{i,k}$ and $\xi_{i,k}$ from (3.29), (3.30), (3.31) and then the gradient values from (3.18)–(3.28).

40: Adjust the parameters according to the iterative formulas in (3.5)–(2.15).

50: Input another learning pattern, go to step 20.

The training examples are presented cyclically until all parameters are stabilized, i.e., until the energy function $E$ for the entire training set is acceptably low and the network converges.

## IV. Universal Approximation Capability of NAHONN

In this section the variables $a1, b1, c1, a2, b2, a3, b3, a4, b4$ are set as piecewise constant functions which will be adjusted (as well as weights) during training.

Now because a mapping

$$f : R^n \to R^m$$

(where $n$ and $m$ are positive integers) can be computed by $m$ mappings

$$f_j : R^n \to R$$

(where $j = 1, 2, \ldots, m$), it is theoretically sufficient to focus on networks with one output-unit only. Such a network is depicted in Fig. 5.

In Fig. 5, the weights-vector and the threshold value associated with the $j$th hidden layer processing unit are denoted $w_j$

and $\Theta_j$, respectively. The weights-vector associated with the single output-unit is denoted $\beta_j$. The input-vector is denoted $x = (x_1, \ldots, x_n)$.

Using these notations, we see that the function computed by an NAHONN is

$$f(x) = \sum_{j=1}^{k} \beta_j \Psi(w_j \cdot x - \Theta_j) \tag{4.1}$$

where $k$ is the number of hidden layer processing units, and $\Psi$ is defined by (3.1).

*Lemma 4.1:* An NAHONN with a neuron-adaptive activation function (3.1) can approximate any piecewise continuous function with infinite (countable) discontinuous points to any degree of accuracy (see Appendix A for proof of this theorem).

## V. NAHONNG

### A. Neural-Network Group

The ANN set is a set in which every element is an ANN [29]. The generalized ANN set—**NN**—is the union of the product set $\mathbf{NN}^*$ and the additive set $\mathbf{NN}^+$.

A nonempty set $\boldsymbol{N}$ is called a *neural-network group*, if $\boldsymbol{N} \subset \mathbf{NN}$ (the generalized neural-network set as described in [12]) with the product $n_i n_j$ or with addition $n_i + n_j$ is defined for every two elements $n_i, n_j \in \boldsymbol{N}$, and for which the group production conditions or group addition conditions hold [29].

An *ANN group set* is a set in which every element is an artificial neural-network group. The symbol $\boldsymbol{N} \in \mathrm{NNGS}$ *means*: NNGS is an ANN group set and $\boldsymbol{N}$, which is one kind of artificial neural-network group, is an element of set NNGS.

In general, the neural-network group set can be written as

$$\mathbf{NNGS} = \{\boldsymbol{N}_1, \boldsymbol{N}_2, \ldots, \boldsymbol{N}_i, \ldots\} \tag{5.1}$$

where $\boldsymbol{N}_i$ is one kind of neural-network group.

### B. NAHONG Neural-Network Group Models

The NAHONG is one kind of neural-network group, in which each element is an NAHONN ($\boldsymbol{Z}_i$) [29]. We have:

$$\boldsymbol{NAHONG} = \{\boldsymbol{Z}_1, \boldsymbol{Z}_2, \boldsymbol{Z}_3, \ldots, \boldsymbol{Z_i}, \ldots, \boldsymbol{Z_n}\} \tag{5.2}$$

### C. NAHONG Neural-Network Group Feature

Hornik proved the following general result:

> "Whenever the activation function is continuous, bounded and nonconstant, then for an arbitrary compact subset $\boldsymbol{X} \subseteq \boldsymbol{R}^n$, standard multilayer feedforward networks can approximate any continuous function on $\boldsymbol{X}$ arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available" [13].

A more general result was proved by Leshno:

> "A standard multilayer feedforward network with a locally bounded piecewise continuous activation function can approximate *any* continuous function to any degree of accuracy if and only if the network's activation function is not a polynomial" [17].

Furthermore, since NAHONG comprises artificial neural networks, we can infer the following from Leshno:

> "Consider a neuron-adaptive higher order neural-network group (NAHONG), in which each element is a standard multilayer higher order neural network with adaptive neurons, and which has locally bounded, piecewise continuous (rather than polynomial) activation function and threshold. Each such group can approximate *any* kind of piecewise continuous function, and to *any* degree of accuracy".

## VI. NANONN and NANONG Examples

### A. T-NAHONN Model

When

$$a2_{i,k} = b2_{i,k} = d2_{i,k} = a3_{i,k} = b3_{i,k} = d3_{i,k} = a4_{i,k} = b4_{i,k} = d4_{i,k} = 0.$$

The 2-D NNAF (2.9) becomes the 2-D trigonometric polynomial NAF

$$\begin{aligned}\Psi_{i,k}(\mathrm{net}_{i,k}) &= \sum_{h=1}^{4} f_{i,k,h}(\mathrm{net}_{i,x,k}, \mathrm{net}_{i,y,k}) \\ &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (\mathrm{net}_{i,x,k})) \\ &\quad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (\mathrm{net}_{i,y,k}))\end{aligned} \tag{6.1}$$

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, d1_{i,k}, e1_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

Let

$$\mathrm{net}_{i,x,k} = w_{i,x,k} \cdot x$$
$$\mathrm{net}_{i,y,k} = w_{i,y,k} \cdot y.$$

The 2-D trigonometric polynomial NAHONN is defined as

T-$NAHONN$ (2-D):

$$\begin{aligned}&\sum_{i=0}^{n} w_{z,i,k+1} \cdot \Psi_{i,k}(\mathrm{net}_{i,x,k}, \mathrm{net}_{i,y,k}) \\ &\quad = \sum_{i=0}^{n} w_{z,i,k+1} \cdot \sum_{h=1}^{3} f_{i,k,h}(w_{i,x,k} \cdot x, w_{i,y,k} \cdot y) \\ &\quad = \sum_{i=0}^{n} w_{z,i,k+1} \cdot ((a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (w_{i,x,k} \cdot x)) \\ &\qquad \cdot \cos^{e1_{i,k}}(d1_{i,k} \cdot (w_{i,y,k} \cdot y)))\end{aligned} \tag{6.2}$$

where

$$a1_{i,k}, b1_{i,k}, c1_{i,k}, d1_{i,k}, e1_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

T-NAHONN is an open-box model, which is capable of handling high-frequency nonlinear data.

### B. PT-NAHONN Model

When

$$a2_{i,k} = b2_{i,k} = d2_{i,k} = a3_{i,k} = b3_{i,k} = d3_{i,k} = 0$$
$$a2_{i,k} = b2_{i,k} = d2_{i,k} = a3_{i,k} = b3_{i,k} = d3_{i,k} = 0$$

the 2-D NAF (2.9) becomes the 2-D polynomial and trigonometric polynomial NAF

$$\begin{aligned}&\Psi_{i,k}(\text{net}_{i,k})\\&\quad=\sum_{h=1}^{4} f_{i,k,h}(\text{net}_{i,x,k},\text{net}_{i,y,k})\\&\quad=a1_{i,k}\cdot\sin^{c1_{i,k}}(b1_{i,k}\cdot(\text{net}_{i,x,k}))\\&\qquad\cdot\cos^{e1_{i,k}}(d1_{i,k}\cdot(\text{net}_{i,y,k}))+a4_{i,k}\cdot(\text{net}_{i,x,k})^{b4_{i,k}}\\&\qquad\cdot(\text{net}_{i,y,k})^{d4_{i,k}}\end{aligned}\tag{6.3}$$

where

$$a1_{i,k},b1_{i,k},c1_{i,k},d1_{i,k},e1_{i,k},a4_{i,k},b4_{i,k},d4_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

Let

$$\begin{aligned}\text{net}_{i,x,k}&=w_{i,x,k}\cdot x\\\text{net}_{i,y,k}&=w_{i,y,k}\cdot y\end{aligned}$$

The 2-D polynomial and trigonometric polynomial NAHONN is defined as

PT-NAHONN (2-D):

$$\begin{aligned}&\sum_{i=0}^{n} w_{z,i,k+1}\cdot\Psi_{i,k}(\text{net}_{i,x,k},\text{net}_{i,y,k})\\&\quad=\sum_{i=0}^{n} w_{z,i,k+1}\cdot\sum_{h=1}^{4} f_{i,k,h}(w_{i,x,k}\cdot x,w_{i,y,k}\cdot y)\\&\quad=\sum_{i=0}^{n} w_{z,i,k+1}\cdot((a1_{i,k}\cdot\sin^{c1_{i,k}}(b1_{i,k}\cdot(w_{i,x,k}\cdot x))\\&\qquad\cdot\cos^{e1_{i,k}}(d1_{i,k}\cdot(w_{i,y,k}\cdot y))+a4_{i,k}\cdot(\text{net}_{i,x,k})^{b4_{i,k}}\\&\qquad\cdot(\text{net}_{i,y,k})^{d4_{i,k}})\end{aligned}\tag{6.4}$$

where

$$a1_{i,k},b1_{i,k},c1_{i,k},d1_{i,k},e1_{i,k},a4_{i,k},b4_{i,k},d4_{i,k}$$

are free parameters which will be adjusted (as well as weights) during training.

PT-NAHONN is appropriate for modeling data which combines both polynomial and trigonometric features.

### C. TES-NAHONN Model

When

$$c1_{i,k}=1,\quad a4_{i,k}=b4_{i,k}=d4_{i,k}=0$$

the 1-D NAF (2.3) becomes the 2-D TES NAF.

The 1-D TES NAF is

$$\begin{aligned}&\Psi_{i,k}(\text{net}_{i,k})\\&\quad=\sum_{h=1}^{4} f_{i,k,h}(\text{net}_{i,k})\\&\quad=a1_{i,k}\cdot\sin(b1_{i,k}\cdot(\text{net}_{i,k}))+a2_{i,k}\cdot e^{-b2_{i,k}\cdot(\text{net}_{i,k})}\\&\qquad+a3_{i,k}\cdot\frac{1}{1+e^{-b3_{i,k}\cdot(\text{net}_{i,k})}}\end{aligned}\tag{6.5}$$

where

$$a1_{i,k},b1_{i,k},a2_{i,k},b2_{i,k},a3_{i,k},b3_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

Let

$$\text{net}_{i,k}=w_{i,x,k}\cdot x$$

The 1-D TES NAHONN is defined as

$$\begin{aligned}&\text{TES- NAHONN (1-D)}=\sum_{i=0}^{n} w_{z,i,k+1}\cdot\Psi_{i,k}(\text{net}_{i,k})\\&\quad=\sum_{i=0}^{n} w_{z,i,k+1}\cdot\sum_{h=1}^{4} f_{i,k,h}(w_{i,x,k}\cdot x)\\&\quad=\sum_{i=0}^{n} w_{z,i,k+1}\cdot\Big(a1_{i,k}\cdot\sin(b1_{i,k}\cdot(w_{i,x,k}\cdot x))+a2_{i,k}\\&\qquad\cdot e^{-b2_{i,k}\cdot(w_{i,x,k}\cdot x)}+a3_{i,k}\cdot\frac{1}{1+e^{-b3_{i,k}\cdot(w_{i,x,k}\cdot x)}}\Big)\end{aligned}\tag{6.6}$$

where

$$a1_{i,k},b1_{i,k},a2_{i,k},b2_{i,k},a3_{i,k},b3_{i,k}$$

are free parameters which can be adjusted (as well as weights) during training.

The TES-NAHONN model is suitable for modeling data comprising discontinuous and piecewise continuous functions.

### D. NAHONG Models

NAHONG is one kind of neural-network group, in which each element is an NAHONN ($\boldsymbol{Z}_i$) [29]. We have

$$\text{NAHONG}=\{\boldsymbol{Z}_1,\boldsymbol{Z}_2,\boldsymbol{Z}_3,\ldots,\boldsymbol{Z}_{\boldsymbol{i}},\ldots,\boldsymbol{Z}_{\boldsymbol{n}}\}$$

If $Z_I=T$PT- NAHONN$_i$, then this becomes the trigonometric polynomial NAHONG.

If $Z_I=$ - NAHONN$_i$, then this becomes the polynomial and trigonometric polynomial NAHONG.

If $Z_I=$ TES-NAHONN$_i$, then this becomes the trigonometric/exponent/sigmoid polynomial NAHONG.

NAHONG models are useful for modeling some special piecewise continuous functions.

The NAHONG structure is shown in Fig. 6.

## VII. Financial Data Modeling Results

### A. T-NAHONN Model Results

*QwikNet* and *Qnet* are two products available for the financial software market (http://www.kagi.com/cjensen/) and (www.qnet97.com), respectively. Both are powerful, yet easy-to-use, ANN simulators. For the purposes of the present study, the authors developed the *MASFinance* software package in order to simulate nonlinear high-frequency real-word data, comprising multipolynomial components. This practical implementation of NAHONN models was developed using

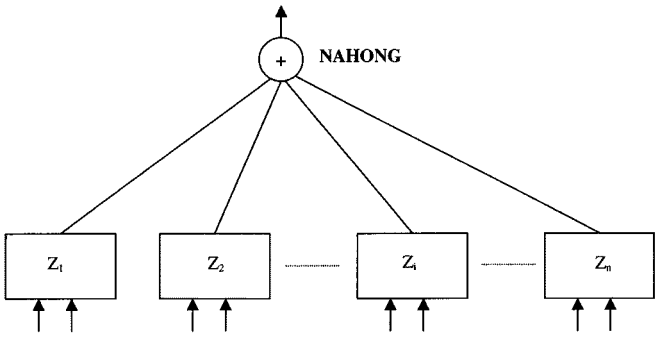


Fig. 6. NAHONG model.

*Xview* under Solaris. The main features of *MASFinance* can be summarized as follows:

*Graphical display of data*. Both training and test data are displayed as 3-D curved surfaces.

*Creating* NAHONNs. The number of layers, number of neurons per layer and neuron type can all be specified (and saved as net files in the system).

*Graphical display of neural networks*. The NAHONN structure can be displayed exhibit graphically, with different neurons (i.e., neurons with different activation functions) marked with different colors; display of weights (connections between neurons) is optional.

*Display of simulation results*. An interpolation algorithm is invoked to smooth simulation results, prior to display.

*Display of learned neuron activation function*—NAF. After training, the NAF parameters are shown in the main window.

*Display of simulation error graph*. The updated average error can be dynamically displayed as training proceeds. For a well-convergent model, the error plot approaches the $x$-axis over time (i.e., the auto-scaled number of epochs).

*Setting other parameters*. Training times and parameters such as learning rate and momentum can be easily set and modified. It only takes 5 min for 100 000 epochs when data points are less than 100. A well-trained model can be saved as a net file, and the output result saved as a performance file.

We now proceed to compare the performance of T-NAHONN, *QwikNet*, and *Qnet*. In order to perform this comparison, we need to establish some ground rules, namely:

- *QwikNet* demonstration data files are used in all three programs. There are eight files in total, these being *IRIS4-1.TRN, IRIS4-3.TRN, SINCOS.TRN, XOR.TRN, XOR4SIG.TRN, SP500.TRN, SPIRALS.TRN* and *SECURITY.TRN*. (*Remark*: it is possible that *QwikNet's* data files have been optimized for that particular program)
- "*Learning Rates*" of all three programs are set to the same value. Usually, the higher the learning rate, the faster the network learns. The valid range is between 0.0 and 1.0. A good initial guess is 0.1 when training a new network. If the learning rate is too high the network may become unstable, at which time the weights should be randomized and training restarted.
- "*Maximum # Epoch*" is set to 100 000.
- The number of "Hidden Layers" is set to one.

TABLE I
COMPARATIVE ANALYSIS SUMMARY (T-NAHONN, *QwikNet* AND *Qnet*)

| Data File | Data Feature | *T-NAHONN* Average Error | *QwikNet* Average Error | *Qnet* Average Error |
|---|---|---|---|---|
| IRIS4-1 | Step | 0.029 | 0.055 | 0.005 |
| IRIS4-3 | Step | 0.028 | 0.071 | 0.001 |
| SINCOS | Linear | 0.011 | 0.038 | 0.005 |
| XOR | XOR | 1.62E-14 | 0.078 | 6.1E-5 |
| XOR4SIG | XOR + nonlinear | 1.62E-14 | 0.037 | 0.350 |
| SP500 | Nonlinear | 0.02 | 0.014 | 0.021 |
| SPIRALS | Nonlinear High Frequency | 0.031 | 0.395 | 0.270 |
| SECURITY | Nonlinear Very High Frequency | 0.018 | 0.045 | 0.023 |

We compared all three programs on each set of input data files independently, with the results being summarized in Table I.

First, comparing T-NAHONN with *QwikNet* reveals that the former yielded smaller simulation errors in the majority of cases (especially so in the case of XOR and *SPIRALS*).

Second, comparing T-NAHONN and *Qnet* reveals similar performance, with both outperforming *QuikNet* (especially so with XOR). These results indicate that polynomial neural networks perform better than trigonometric neural networks for this kind of input data.

In summary, the T-NAHONN model works well for nonlinear, high frequency, and higher order functions. The reason for this is that T-NAHONN is able to *automatically* determine the optimum order (either integer or real number) for such high frequency data.

### B. PT-NAHONN Results

The network architecture of PT-NAHONN combines both the characteristics of PHONN [27] and THONN [25], [26], [32]. It is a multilayer network that consists of an input layer with input-units, output layer with output-units, and two hidden layers consisting of intermediate processing units. After the PT-HONN simulator was developed, tests were undertaken to prove it worked well with *real-world* data. Subsequently, its performance was compared with other artificial neural-network models. More specifically, comparative financial data simulation results were derived using Polynomial Higher Order Neural Network PT-NAHONN, PHONN, and Trigonometric polynomial Higher Order neural-network (THONN). Particular attention is paid here on the degree of the accuracy of the error.

Once again, certain ground rules were adopted to ensure testing consistency, namely:

- programs were operated within the same environment;
- average percent error was the metric used for comparison purposes;
- the same test data and environmental parameters (such as learning rate and number of hidden layers) were used throughout.

Representative results obtained are shown in Table II. From these results we see that the PT-NAHONN model always performs better than both PHONN and THONN. For example,

TABLE II
PT-NAHONN, PHONN, AND THONN COMPARISON RESULTS

| *Data* | *PHONN Average \|error\|%* | *THONN Average \|error\|%* | *PT-NAHONN Average \|error\|%* |
|---|---|---|---|
| *All Banks - Liabilities and Assets on NonResidents of Total Liabilities between 96.8- 97.5 (Reserve Bank of Australia Bulletin 1999.7 Page S3)* | *1.9895%* | *2.0976%* | *1.9375%* |
| *All Banks – Credit Card Lending on Total Number of Accounts between 96.8- 97.6 (Reserve Bank of Australia Bulletin 1999.7 P. S18)* | *1.8927%* | *0.9717%* | *0.5208%* |
| Exchange Rate on Japanese (yen) between *96.8 - 97.6 (Reserve Bank of Australia Bulletin 1999.7 Page S53)* | *2.3452%* | *2.2780%* | *0.9130%* |
| Exchange Rate on United States (dollar) *between 96.8 - 97.6 (Reserve Bank of Australia Bulletin 1999.7 Page S53)* | *1.6057%* | *1.6969%* | *1.3283%* |
| Labour Force on Labour *between 97.8 – 98.6 (Reserve Bank of Australia Bulletin 1999.7 Page S63)* | *4.0252%* | *5.3549%* | *3.9827%* |
| Share Price Indices – United States on Dow Jones industrial *between 1998.8- 99.6 (Reserve Bank of Australia Bulletin 1999.7 Page S50)* | *3.5730%* | *3.5755%* | *3.4853%* |

Using All Banks-Credit Card Lending on Total Number of Accounts data between 96.8–97.6 (Reserve Bank of Australia Bulletin 1999.7 P. S18), the error of the PT-NAHONN model was 0.5208%. By comparison, the PHONN error was 1.8927%, and the THONN error 0.9717%. As it happens, the All Banks-Credit Card Lending on Total Number of Accounts data is a mixture of polynomial and trigonometric polynomial. The PHONN model can only simulate polynomial functions well, whereas THONN is only able to simulate trigonometric functions well. Moreover, PT-NAHONN *automatically* finds the optimum model (in this case, combined polynomial and trigonometric polynomial function), and correct order (real number order), for modeling this data.

### *C. Finding the Best Model and Order for Financial Data Using TES-NAHONN*

Our next experiment was to simulate nonlinear discontinuous financial data using NAHONN with NAF (7.3.1) as the neuron activation function. Reserve Bank of Australia data (Lending to Persons July 1994 to Dec 1996) is shown in Fig. 7. We observe that there is a sudden jump between 06/95 and 07/95 (from 32.383 to 36.464). We therefore assume that discontinuous points exist here, and accordingly divide the entire time series into two continuous portions.

Let us simulate this piecewise continuous time series using a single NAHONN with piecewise NAF. There are 30 data points in the training data set in this particular sample. After training, the learned NAF for the NAHONN becomes

$$\Psi(x) = A1 \cdot \sin(B1 \cdot x) + A2 \cdot e^{-B2 \cdot x} + \frac{A3}{1 + e^{-B3 \cdot x}} \qquad x \in [1, 12] \cup [13, 30] \quad (7.3.1)$$

where

$$A1 = \begin{cases} 0.04, & x \in [1, 12] \\ 12.56, & x \in [13, 30] \end{cases}$$

$$B1 = \begin{cases} 3.25, & x \in [1, 12] \\ -0.99, & x \in [13, 30] \end{cases}$$

$$A2 = \begin{cases} 5.33, & x \in [1, 12] \\ -0.09, & x \in [13, 30] \end{cases}$$

$$B2 = \begin{cases} -3.56, & x \in [1, 12] \\ 8.21, & x \in [13, 30] \end{cases}$$

$$A3 = \begin{cases} 2.15, & x \in [1, 12] \\ 5.42, & x \in [13, 30] \end{cases}$$

$$B3 = \begin{cases} -6.88, & x \in [1, 12] \\ 1.42, & x \in [13, 30]. \end{cases}$$

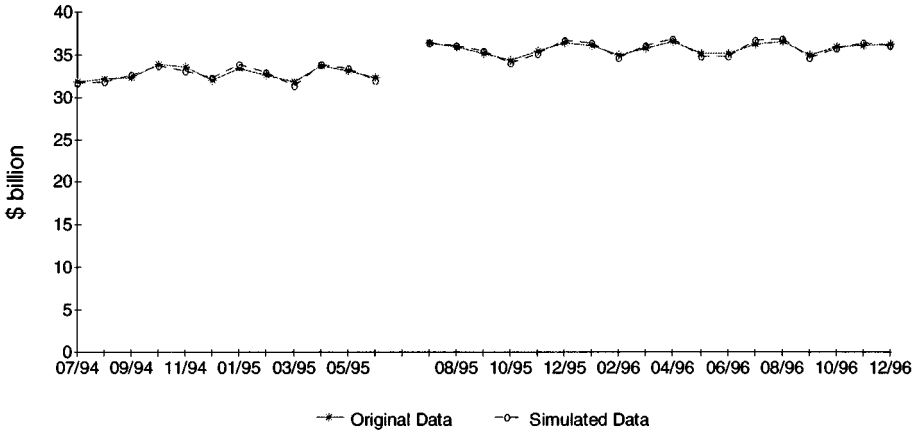


Fig. 7. Single NAHONN with piecewise NAF used to simulate discontinuous data of Reserve Bank of Australia lending to persons July 1994 to December 1996 (Reserve Bank of Australia Bulletin, 1997).

TABLE III
COMPARISON OF SINGLE TES-NAHONN WITH THONN AND PHONN

| Higher-Order Neural Network Model | # Data Points | # Hidden Layer Neurons | # Epochs | RMS Error |
|---|---|---|---|---|
| *TES-NAHONN* with piecewise NAF | 30 | 4 | 6,000 | 0.01513 |
| *TES-NAHONN* with nonlinear NAF | 30 | 9 | 20,000 | 0.76497 |
| *THONN* | 30 | 9 | 20,000 | 1.13525 |
| *PHONN* | 30 | 9 | 20,000 | 1.20193 |

With only four hidden processing units and 6000 training epochs, this NAHONN with piecewise NAF reaches a root-mean-square (rms) error of only 0.015 31 (see Table III).

We also tried approximating this financial data set using a single NAHONN with nonlinear continuous NAF. After training, the learned nonlinear NAF for the TES-NAHONN becomes

$$\Psi(x) = 0.09\sin(1.73x) - 5.06e^{12.96x} + \frac{2.33}{1+e^{-1.21x}}. \tag{7.3.2}$$

With nine hidden units and 20 000 training epochs, this TES-NANONN with nonlinear NAF reached an rms error of 0.764 97.

With the same numbers of hidden units and training epochs, the rms errors for PHONN and THONN were 1.201 93 and 1.135 25, respectively. Using the TES-NAHONN model, we found that (7.3.2) best modeled the data, provided we use a nonlinear continuous NAF. Moreover, the optimum order was 12.96 for exponent and $-1.21$ for sigmoid function. Using PHONN, we would need to choose a polynomial model with *integer* order. Thus PHONN cannot model such data very well. Similarly, using THONN means that the trigonometric polynomial model must also use an integer number, resulting in inferior performance. Our results show that the best model for this data must be a mixture of trigonometric, exponential, and sigmoid functions (for other data, variable a1 in (2.1) might be zero, and the trigonometric function would not be chosen). Furthermore, the orders are *real* numbers. In summary, TES-NAHONN can adjust the activation function and automatically chose the best model and correct order for financial modeling.

### D. NAHONG Results

In the following experiments we consider three kinds of neural-network group, and compare their approximation properties:

- traditional fixed sigmoid FNN group (FNNG) in which each element is a traditional FNN [28];
- Neuron-adaptive FNN group (NAFG) in which each element is a neuron-adaptive FNN [31];
- NAHONG.

Some special functions (such as the Gamma-function) possess significant properties and find application in engineering and physics. These special functions, defined by integration or power series, exist as the solutions to some differential equations that emerge from practical problems, and cannot be

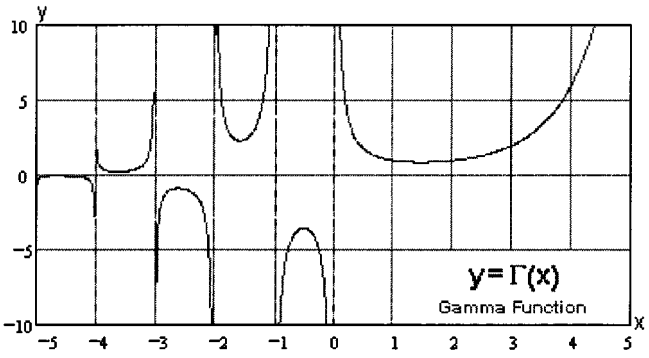


Fig. 8. Gamma-function.

expressed by elementary functions. As the Gamma-function is basically a piecewise continuous function, in this section we attempt to approximate it using neural-network groups. The Gamma-function (with real variables) is defined as in Fig. 8

$$\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}\,dt$$
$$\Gamma(x+1) = \int_0^\infty e^{-t}t^{x}\,dt = [-e^{-t}t^{x}]_0^\infty + x\int_0^\infty e^{-t}t^{x-1}\,dt$$
$$= x\int_0^\infty e^{-t}t^{x-1}\,dt = x\Gamma(x).$$

It is readily seen from Fig. 8 that the Gamma-function is a piecewise continuous function and therefore can be approximated to any degree of accuracy using neural-network groups. For the sake of simplicity, we only consider the above function over the range $[-2, 4]$, i.e., three continuous parts.

The best approximation result using NAHONG was achieved with 3000 epochs of training and two hidden units in each neuron-adaptive higher order neural network. The learned NAFs for the hidden layers are (Figs. 9–11)

$$\sigma_1(x) = 4.35\sin(-1.77x) + 1.16e^{-9.03x} + \frac{0.05}{1+e^{-0.66x}}, \quad x, y \in (-2, -1)$$
$$\sigma_2(x) = 6.59\sin(1.82x) - 0.61e^{-8.23x} + \frac{4.28}{1+e^{-0.96x}}, \quad x, y \in (-1, 0)$$
$$\sigma_3(x) = 9.12\sin(0.28x) - 3.33e^{-1.46x} + \frac{0.51}{1+e^{-1.98x}} \quad x, y \in (0, 4).$$

A comparison between FNN group, NAF group, and NAHONN groups is given in the Table IV, from which we see that the overall rms error for the NAHONG was only 0.001 324.

The results of our experiments confirm that NAHONGs possess the most outstanding approximation properties—namely fastest learning, greatly reduced network size, and most accurate fitting for piecewise functions, compared with FNNG and NAFG. The reason for this is that NAHONG can not only automatically find the best model and correct order, but can also model higher frequency, multipolynomial, discontinuous, and piecewise functions.

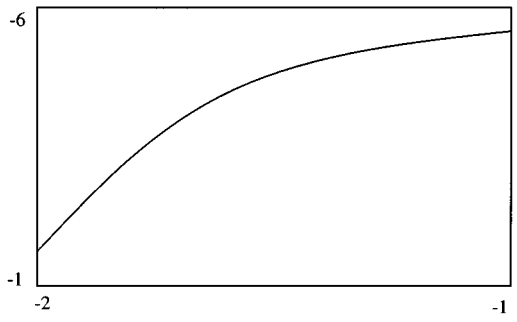


Fig. 9. The learned NAF $\sigma_1(x)$.

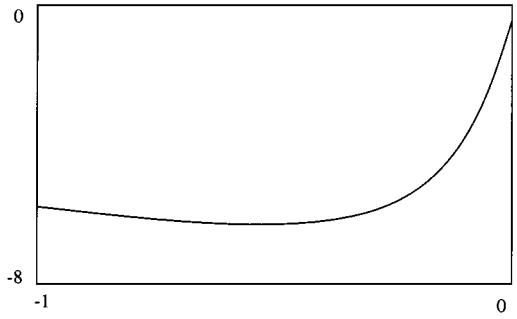


Fig. 10. The learned NAF $\sigma_2(x)$.

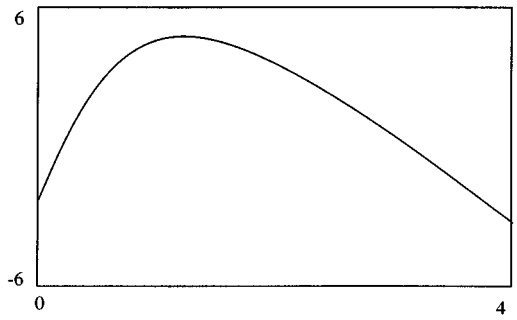


Fig. 11. The learned NAF $\sigma_3(x)$.

TABLE IV
FNN GROUP, NAF GROUP, AND NAHONN GROUP COMPARISON

| Model | Hidden Unites | Epochs | RMS |
|---|---|---|---|
| *FNNG* | 12 | 30,000 | 0.986034 |
| *NAFG* | 4 | 5,000 | 0.009327 |
| *NAHONG* | 2 | 3,000 | 0.001324 |

## VIII. CONCLUSION

In this paper, we have presented NAHONN models with neuron-adaptive activation functions to automatically model any continuous function (or any piecewise continuous function), to any desired accuracy. A learning algorithm was derived

to tune both the free parameters in the neuron-adaptive activation function, as well as the connection weights between the neurons. We proved that a single NAHONN can approximate any piecewise continuous function, to any desired accuracy. Experiments with function approximation and financial data simulation indicate that the proposed NAHONN models offer significant advantages over traditional neural networks (including much reduced network size, faster learning, and smaller simulation error). Our experiments also suggest that a single NAHONN model can effectively approximate piecewise continuous functions, and offer superior performance compared with neural-network groups.

We further introduced specialized NAHONN models, including trigonometric polynomial NAHONN (T-NAHONN), polynomial and trigonometric polynomial NAHONN (PT-NAHONN), trigonometric/exponent/sigmoid polynomial NAHONN (TES-NAHONN), and NAHONN group (NAHONG) models. These NAHONN and NAHONG models are open box, convergent models, are able to handle high-frequency data, model multipolynomial data, simulate discontinuous data, and are capable of approximating any kind of piecewise continuous function, to any degree of accuracy. Experimental results obtained using a NAHONN and NAHONG were presented, which confirm that both models are also capable of automatically finding the optimum model and appropriate order for financial data modeling.

Having defined neuron-adaptive higher order neural networks, there remains considerable scope for further characterization and development of appropriate algorithms. Research into *hybrid* neural-network group models also represents a promising avenue for future investigation. Neuron-adaptive higher order neural-network models hold considerable promise for both the understanding and development of complex financial systems. Neuron-adaptive higher order neural-network group research remains open-ended, and holds considerable potential for developing neural-network group-based models for complex financial systems.

## APPENDIX A

*Proof of Lemma 4.1:* Let us first consider a piecewise discontinuous function $f(x)$ defined on a compact set $S \subset R^n$ and with only one discontinuous point, which divides $f(x)$ into two continuous parts: $f_1(x)$ and $f_2(x)$, i.e., (see (4.2) at the bottom of the page). Now, each of the continuous parts can be approximated (to any desired accuracy) by an NAHONN (according to Leshno *et al.* [17]). In other words, for $f_1(x)$, there exist weight—vector $w_{1,j}$, weights $\beta_{1,j}$, thresholds $\theta_{1,j}$, and real numbers $a1_{1,j}, a2_{1,j}, a3_{1,j}, a4_{1,j}, b1_{1,j}, b2_{1,j}, b3_{1,j}, b4_{1,j}, c1_{1,j}$ such that

$$\begin{aligned}\hat{f}_1(x) &= \sum_{j=1}^{k_1} \beta_{1,j}\Psi(w_{1,j}\cdot x-\theta_{1,j}) \\ &= \sum_{j=1}^{k_1} \beta_{1,j}\Big[a1_{1,j}\cdot \sin^{c1_{1,j}}(b1_{1,j}\cdot(w_{1,j}\cdot x-\theta_{1,j})) \\ &\quad + a2_{1,j}\cdot e^{-b2_{1,j}\cdot(w_{1,j}\cdot x-\theta_{1,j})^2} \\ &\quad + \frac{a3_{1,j}}{1+e^{-b3_{1,j}\cdot(w_{1,j}\cdot x-\theta_{1,j})}} \\ &\quad + a4_{1,j}(w_{1,j}\cdot x-\theta_{1,j})^{b4_{1,j}}\Big] \\ &= \sum_{j=1}^{k_1}\Big[\beta_{1,j}\cdot a1_{1,j}\cdot \sin^{c1_{1,j}}(b1_{1,j}\cdot w_{1,j}\cdot x - b1_{1,j} \\ &\quad \cdot\theta_{1,j}) + \beta_{1,j}\cdot a2_{1,j}\cdot e^{-b2_{1,j}(w_{1,j}\cdot x-\theta_{1,j})^2} \\ &\quad + \frac{\beta_{1,j}\cdot a3_{1,j}}{1+e^{-(b3_{1,j}\cdot w_{1,j}\cdot x-b3_{1,j}\cdot\theta_{1,j})}} \\ &\quad + \beta_{1,j}\cdot a4_{1,j}(w_{1,j}\cdot x-\theta_{1,j})^{b4_{1,j}}\Big] \end{aligned} \tag{A.1}$$

is a desired approximation to $f_1(x), x \in S_1$, where $k_1$ is the required number of hidden units and $\Psi$ is the (piecewise) neuron adaptive activation function.

On the other hand, for $f_2(x)$, there exist weight vector $w_{2,j}$, weights $\beta_{2,j}$, thresholds $\theta_{2,j}$, and real numbers $a1_{2,j}, a2_{2,j}, a3_{2,j}, a4_{2,j}, b1_{2,j}, b2_{2,j}, b3_{2,j}, b4_{2,j}, c1_{2,j}$ such that

$$\begin{aligned}\hat{f}_2(x) &= \sum_{j=1}^{k_2} \beta_{2,j}\Psi(w_{2,j}\cdot x-\theta_{2,j}) \\ &= \sum_{j=1}^{k_2} \beta_{2,j}\Big[a1_{2,j}\cdot \sin^{c1_{2,j}}(b1_{2,j}\cdot(w_{2,j}\cdot x-\theta_{2,j})) \\ &\quad + a2_{2,j}\cdot e^{-b2_{2,j}\cdot(w_{2,j}\cdot x-\theta_{2,j})^2} \\ &\quad + \frac{a3_{2,j}}{1+e^{-b3_{2,j}\cdot(w_{2,j}\cdot x-\theta_{2,j})}} \\ &\quad + a4_{2,j}(w_{2,j}\cdot x-\theta_{2,j})^{b4_{2,j}}\Big] \\ &= \sum_{j=1}^{k_2}\Big[\beta_{2,j}\cdot a1_{2,j}\cdot \sin^{c1_{1,j}}(b1_{2,j}\cdot w_{2,j}\cdot x - b1_{2,j} \\ &\quad \cdot\theta_{2,j}) + \beta_{2,j}\cdot a2_{2,j}\cdot e^{-b2_{2,j}(w_{2,j}\cdot x-\theta_{2,j})^2} \\ &\quad + \frac{\beta_{2,j}\cdot a3_{2,j}}{1+e^{-(b3_{2,j}\cdot w_{2,j}\cdot x-b3_{2,j}\cdot\theta_{2,j})}} \\ &\quad + \beta_{2,j}\cdot a4_{2,j}(w_{2,j}\cdot x-\theta_{2,j})^{b4_{2,j}}\Big] \end{aligned} \tag{A.2}$$

$$f(x) = \begin{cases} f_1(x), & x \in S_1 \subset R^n \\ f_2(x), & x \in S_2 \subset R^n \end{cases}, \quad S_1 \cup S_2 = S, \quad S_1 \cap S_2 = \phi. \tag{4.2}$$

where $k_2$ is the required number of hidden units and $\Psi$ is the (piecewise) neuron adaptive activation function, is a desired approximant to $f_2(x), x \in S_2$.

Now our aim is to prove that both $f_1(x)$ and $f_2(x)$ can be approximated arbitrarily well using a single neural network. In order to do this, we rewrite $\hat{f}_2(x)$ as

$$\begin{aligned}\hat{f}_2(x) = \sum_{j=1}^{k_2} \beta_{1,j} \Bigg[ & \frac{\beta_{2,j} \cdot a1_{2,j}}{\beta_{1,j}} \\ & \cdot \sin^{c1_{2,j}} \left( \frac{b1_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \cdot w_{1,j} \cdot x - b1_{2,j} \cdot \theta_{2,j} \right) \\ & + \frac{\beta_{2,j} \cdot a2_{2,j}}{\beta_{1,j}} \cdot e^{-\left( \frac{b2_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \cdot w_{1,j} \cdot x - b2_{2,j} \cdot \theta_{2,j} \right)} \\ & + \frac{\beta_{2,j} \cdot a3_{2,j} / \beta_{1,j}}{1 + e^{-\left( \frac{b3_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \cdot w_{1,j} \cdot x - b3_{2,j} \cdot \theta_{2,j} \right)}} \\ & + \frac{\beta_{2,j} \cdot a4_{2,j}}{\beta_{1,j}} \left( \frac{w_{2,j} \cdot x}{w_{1,j} \cdot x} \cdot w_{1,j} \cdot x - \theta_{2,j} \right)^{b4_{2,j}} \Bigg]. \end{aligned} \tag{A.3}$$

If we let

$$\begin{aligned} a1'_{2,j} &= \frac{\beta_{2,j} \cdot a1_{2,j}}{\beta_{1,j}}, \quad b1'_{2,j} = \frac{b1_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \\ a2'_{2,j} &= \frac{\beta_{2,j} \cdot a2_{2,j}}{\beta_{1,j}}, \quad b2'_{2,j} = \frac{b2_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \\ a3'_{2,j} &= \frac{\beta_{2,j} \cdot a3_{2,j}}{\beta_{1,j}}, \quad b3'_{2,j} = \frac{b3_{2,j} \cdot w_{2,j} \cdot x}{w_{1,j} \cdot x} \\ \theta'_{2,j} &= \frac{w_{1,j} \cdot x}{w_{2,j} \cdot x} \cdot \theta_{2,j}, \quad a4'_{2,j} = \frac{\beta_{2,j} \cdot a4_{2,j}}{\beta_{1,j}} \\ b4'_{2,j} &= b4_{2,j}, \quad c1'_{2,j} = c1_{2,j} \end{aligned}$$

then (A.3) can be rewritten as

$$\begin{aligned}\hat{f}_2(x) = \sum_{j=1}^{k_2} \beta_{1,j} \Big[ & a1'_{2,j} \cdot \sin^{c1'_{2,j}} b1'_{2,j} (w_{1,j} \cdot x - \theta'_{2,j}) \\ & + a2'_{2,j} \cdot e^{-b2'_{2,j} (w_{1,j} \cdot x - \theta'_{2,j})^2} \\ & + \frac{a3'_{2,j}}{1 + e^{-b3'_{2,j} (w_{1,j} \cdot x - \theta'_{2,j})}} \\ & + a4'_{2,j} (w_{1,j} \cdot x - \theta'_{2,j})^{b4'_{2,j}} \Big]. \end{aligned} \tag{A.4}$$

Suppose that $k_1 > k_2$ (without loss of generality), if we define

$$A1_j = \begin{cases} a1_{1,j}, & x \in S_1 \\ a1'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$B1_j = \begin{cases} b1_{1,j}, & x \in S_1 \\ b1'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$A2_j = \begin{cases} a2_{1,j}, & x \in S_1 \\ a2_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$B2_j = \begin{cases} b2_{1,j}, & x \in S_1 \\ b2'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$A3_j = \begin{cases} a3_{1,j}, & x \in S_1 \\ a3'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$B3_j = \begin{cases} b3_{1,j}, & x \in S_1 \\ b3'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$A4_j = \begin{cases} a4_{1,j}, & x \in S_1 \\ a4'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$B4_j = \begin{cases} b4_{1,j}, & x \in S_1 \\ b4'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$\Theta_j = \begin{cases} \theta_{1,j}, & x \in S_1 \\ \theta'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

$$C1_j = \begin{cases} c1_{1,j}, & x \in S_1 \\ c2'_{2,j}, & x \in S_2 \\ 0, & x \in S_2, \quad j = k_1 - k_2 \end{cases}$$

then (A.2) and (A.4) can be combined as

$$\begin{aligned}\hat{f}(x) &= \sum_{j=1}^{k} \beta_{1,j} \Big[ A1_j \cdot \sin^{C1_j} B1_j (w_{1,j} \cdot x - \Theta_j) \\ &\quad + A2_j \cdot e^{-B2_j (w_{1,j} \cdot x - \Theta_j)^2} + \frac{A3_j}{1 + e^{-B3_j (w_{1,j} \cdot x - \Theta_j)}} \Big] \\ &\quad + A4_j (w_{1,j} \cdot x - \Theta_j)^{B4_j} \\ &= \sum_{j=1}^{k} \beta_{1,j} \Psi (w_{1,j} \cdot x - \Theta_j) \end{aligned} \tag{A.5}$$

where $x \in S$ and $k = k_1$.

It is obvious that (A.5) is a desired approximation to $f(x)$, and that it represents an NAHONN with $k$ hidden units, weight vector $w_{1,j}$, weights $\beta_{1,j}$, thresholds $\Theta_j, j = 1, \ldots, k$, and piecewise adaptive activation function $\Psi$. This demonstrates that a piecewise continuous function with one discontinuous point can be approximated arbitrarily well by an NAHONN.

Next, suppose that a piecewise continuous function $f_m(x)$ ($x \in$ a compact set $S_m \subset R^n$) with $m$ discontinuous points can be arbitrarily approximated by an NAHONN. We proceed to prove that a piecewise continuous function with $m+1$ discontinuous points

$$f(x) = \begin{cases} f_m(x), & x \in S_m \subset R^n \\ f_{m+1}(x), & x \in S_{m+1} \subset R^n \end{cases}$$

can also be approximated to any desired accuracy by an NAHONN.

Suppose that $f_m(x)$ can be approximated by an NAHONN with $k_m$ hidden units, weight vector $w_{m,j}$, weights $\beta_{m,j}$, thresholds $\Theta_{m,j}$, and piecewise adaptive activation function $\Psi_m$ with piecewise constant functions $A1_{m,j}, B1_{m,j}, A2_{m,j}, B2_{m,j}, A3_{m,j}, B3_{m,j}, A4_{m,j}, B4_{m,j}, C1_{m,j}, j = 1, \ldots, k_m$. In other words, $f_m(x)$ can be approximated to any degree of accuracy by

$$\begin{aligned}\hat{f}_m(x) &= \sum_{j=1}^{k_m} \beta_{m,j} \Psi_m(w_{m,j} \cdot x - \Theta_{m,j}) \\ &= \sum_{j=1}^{k_m} \beta_{m,j} \left[ A1_{m,j} \cdot \sin^{C1_{m,j}} B1_{m,j}(w_{m,j} \right. \\ &\quad \cdot x - \Theta_{m,j}) + A2_{m,j} \cdot e^{-B2_{m,j}(w_{m,j} \cdot x - \Theta_{m,j})^2} \\ &\quad \left. + \frac{A3_{m,j}}{1 + e^{-B3_{m,j}(w_{m,j} \cdot x - \Theta_{m,j})}} \right] \\ &\quad + A4_{m,j}(w_{m,j} \cdot x - \Theta_{m,j})_{m,j}^{B4}. \end{aligned} \tag{A.6}$$

On the other hand, $f_{m+1}(x)$ can be approximated arbitrarily well by an NAHONN. In other words, for $f_{m+1}(x)$ there exists weight vector $w_{m+1,j}$, weights $\beta_{m+1,j}$, thresholds $\theta_{m+1,j}$, and real numbers $a1_{m+1,j}, a2_{m+1,j}, a3_{m+1,j}, b1_{m+1,j}, b2_{m+1,j}, b3_{m+1,j}, a4_{m+1,j}, b4_{m+1,j}, c1_{m+1,j}$ such that

$$\begin{aligned}\hat{f}_{m+1}(x) &= \sum_{j=1}^{k_{m+1}} \beta_{m+1,j} \Psi(w_{m+1,j} \cdot x - \theta_{m+1,j}) \\ &= \sum_{j=1}^{k_{m+1}} \beta_{m+1,j} \left[ a1_{m+1,j} \cdot \sin^{c1_{m+1,j}}(b1_{m+1,j} \right. \\ &\quad \cdot (w_{m+1,j} \cdot x - \theta_{m+1,j})) \\ &\quad + a2_{m+1,j} \cdot e^{-b2_{m+1,j} \cdot (w_{m+1,j} \cdot x - \theta_{m+1,j})^2} \\ &\quad + \frac{a3_{m+1,j}}{1 + e^{-b3_{m+1,j} \cdot (w_{m+1,j} \cdot x - \theta_{m+1,j})}} \\ &\quad \left. + a4_{m+1,j}(w_{m+1,j} \cdot x - \theta_{m+1,j})^{b4_{m+1,j}} \right] \\ &= \sum_{j=1}^{k_{m+1}} \left[ \beta_{m+1,j} \cdot a1_{m+1,j} \cdot \sin^{c1_{m+1,j}} \right. \\ &\quad \times (b1_{m+1,j} \cdot w_{m+1,j} \cdot x - b1_{m+1,j} \cdot \theta_{m+1,j}) \\ &\quad + \beta_{m+1,j} \cdot a2_{m+1,j} \cdot e^{-b2_{m+1,j}(w_{m+1,j} \cdot x - \theta_{m+1,j})^2} \\ &\quad + \frac{\beta_{m+1,j} \cdot a3_{m+1,j}}{1 + e^{-(b3_{m+1,j} \cdot w_{m+1,j} \cdot x - b3_{m+1,j} \cdot \theta_{m+1,j})}} \\ &\quad \left. + \beta_{m+1,j} a4_{m+1,j}(w_{m+1,j} \cdot x - \theta_{m+1,j})^{b4_{m+1,j}} \right] \end{aligned} \tag{A.7}$$

where $k_{m+1}$ is the required number of hidden units and $\Psi$ is the neuron adaptive activation function, is a desired approximant to $f_{m+1}(x), x \in S_{m+1}$.

Our aim is to prove that both $f_m(x)$ and $f_{m+1}(x)$ can be approximated using a single neural network. To do this, let us rewrite $\hat{f}_{m+1}(x)$ as

$$\begin{aligned}\hat{f}_{m+1}(x) &= \sum_{j=1}^{k_{m+1}} \beta_{m,j} \left[ \frac{\beta_{m+1,j} \cdot a1_{m+1,j}}{\beta_{m,j}} \cdot \sin^{c1_{m+1,j}} \right. \\ &\quad \times \left( \frac{b1_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \cdot w_{m,j} \cdot x - b1_{m+1,j} \right. \\ &\quad \left. \cdot \theta_{m+1,j} \right) \frac{\beta_{m+1,j} \cdot a2_{m+1,j}}{\beta_{m,j}} \\ &\quad \cdot e^{-(\frac{b2_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \cdot w_{m,j} \cdot x - b2_{m+1,j} \cdot \theta_{m+1,j})} \\ &\quad + \frac{\beta_{m+1,j} \cdot a3_{m+1,j} / \beta_{m,j}}{1 + e^{-(\frac{b3_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \cdot w_{m,j} \cdot x - b3_{m+1,j} \cdot \theta_{m+1,j})}} \\ &\quad + \frac{\beta_{m+1,j} \cdot a4_{m+1,j}}{\beta_{m,j}} \\ &\quad \left. \times \left( \frac{w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \cdot w_{m,j} \cdot x - \theta_{m+1,j} \right)^{b4_{m+1,j}} \right]. \end{aligned} \tag{A.9}$$

If we define

$$\begin{aligned} a1'_{m+1,j} &= \frac{\beta_{m+1,j} \cdot a1_{m+1,j}}{\beta_{m,j}} \\ b1'_{m+1,j} &= \frac{b1_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \\ a2'_{m+1,j} &= \frac{\beta_{m+1,j} \cdot a2_{m+1,j}}{\beta_{m,j}} \\ b2'_{m+1,j} &= \frac{b2_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \\ a3'_{m+1,j} &= \frac{\beta_{m+1,j} \cdot a3_{m+1,j}}{\beta_{m,j}} \\ b3'_{m+1,j} &= \frac{b3_{m+1,j} \cdot w_{m+1,j} \cdot x}{w_{m,j} \cdot x} \\ a4'_{m+1,j} &= \frac{\beta_{m+1,j} \cdot a4_{m+1,j}}{\beta_{m,j}} \\ b4'_{m+1,j} &= b4_{m+1,j} \\ \theta'_{m+1,j} &= \frac{w_{m,j} \cdot x}{w_{m+1,j} \cdot x} \cdot \theta_{m+1,j} \\ c1'_{m+1,j} &= c1_{m+1,j} \end{aligned}$$

then (A.8) can be rewritten as

$$\begin{aligned}\hat{f}_{m+1}(x) &= \sum_{j=1}^{k_{m+1}} \beta_{m,j} \left[ a1'_{m+1,j} \cdot \sin^{c1_{m+1,j}} b1'_{m+1,j} \right. \\ &\quad \times (w_{m,j} \cdot x - \theta'_{m+1,j}) \\ &\quad + a2'_{m+1,j} \cdot e^{-b2'_{m+1,j}(w_{m,j} \cdot x - \theta'_{m+1,j})^2} \\ &\quad + \frac{a3'_{m+1,j}}{1 + e^{-b3'_{m+1,j}(w_{m,j} \cdot x - \theta'_{m+1,j})}} \\ &\quad \left. + a4'_{m+1,j}(w_{m,j} \cdot x - \theta'_{m+1,j})^{b4'_{m+1,j}} \right]. \end{aligned} \tag{A.9}$$

Now, if $k_m > k_{m+1}$, we define

$$A1_j = \begin{cases} A1_{m,j}, & x \in S_m \\ a1'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$B1_j = \begin{cases} B1_{m,j}, & x \in S_m \\ b1'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$A2_j = \begin{cases} A2_{m,j}, & x \in S_m \\ a2'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$B2_j = \begin{cases} B2_{m,j}, & x \in S_m \\ b2'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$A3_j = \begin{cases} A3_{m,j}, & x \in S_m \\ a3'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$B3_j = \begin{cases} B3_{m,j}, & x \in S_m \\ b3'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$A4_j = \begin{cases} A4_{m,j}, & x \in S_m \\ a4'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$B4_j = \begin{cases} B4_{m,j}, & x \in S_m \\ b4'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$\Theta_j = \begin{cases} \Theta_{m,j}, & x \in S_m \\ \theta'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

$$C1_j = \begin{cases} C1_{m,j}, & x \in S_m \\ c1'_{m+1,j}, & x \in S_{m+1} \\ 0, & x \in S_{m+1}, \quad j = k_m - k_{m+1} \end{cases}$$

(In the case of $k_m < k_{m+1}$, set $A1_j, B1_j, A2_j, B2_j, A3_j, B3_j, A4_j, B4_j, C1_j, \Theta_j$ all to zero when $j > \min(k1, k2)$ and $x \in S_m$.) Then (A.6) and (A.9) can be combined as

$$\begin{aligned} \hat{f}(x) &= \sum_{j=1}^{k} \beta_{m,j} \Big[ A1_j \cdot \sin^{C1_j} B1_j (w_{m,j} \cdot x - \Theta_j) \\ &\quad + A2_j \cdot e^{-B2_j (w_{m,j} \cdot x - \Theta_j)^2} + \frac{A3_j}{1 + e^{-B3_j (w_{m,j} \cdot x - \Theta_j)}} \\ &\quad + A4_j (w_{m,j} \cdot x - \Theta_j)^{B4_j} \Big] \\ &= \sum_{j=1}^{k} \beta_{m,j} \Psi (w_{m,j} \cdot x - \Theta_j). \end{aligned} \tag{A.10}$$

where $x \in S = S_m \cup S_{m+1}$ and $k = \max(k_m, k_{m+1})$.

It can be readily seen that (A.10) is a desired approximation to $f(x)$, and that it represents an NAHONN with $k$ hidden units, weight vector $w_{m,j}$, weights $\beta_{m,j}$, thresholds $\Theta_j, j = 1, \ldots, k$, and piecewise adaptive activation function $\Psi$. This demonstartes that a piecewise continuous function with $m+1$ discontinuous points can be approximated arbitrarily well by an NAHONN. Thus, by induction, this completes the proof of Lemma 4.1.

## ACKNOWLEDGMENT

The authors would like to thank J. C. Zhang and B. Lu for their contributions in obtaining some of the experimental results.

## REFERENCES

[1] M. Azema-Barac and A. Refenes, *Handbook of Neural Computation*, E. Fiesler and R. Beale, Eds. Oxford, U.K.: Oxford Univ. Press, 1997, ch. G6.3. in Neural Networks for Financial Applications.

[2] E. Azoff, *Neural Network Time Series Forecasting of Financial Markets*. New York: Wiley, 1994.

[3] A. Barron, "Universal approximation bounds for superposition of a sigmoidal function," *IEEE Trans. Inform. Theory*, vol. 3, pp. 930–945, 1993.

[4] E. Blum and K. Li, "Approximation theory and feedforward networks," *Neural Networks*, vol. 4, pp. 511–515, 1991.

[5] T. Burns, "The interpretation and use of economic predictions," in *Proc. Roy. Soc. A*, 1986, pp. 103–125.

[6] P. Campolucci, F. Capparelli, S. Guarnieri, F. Piazza, and A. Uncini, "Neural networks with adaptive spline activation function," *Proc. IEEE MELECON96*, pp. 1442–1445, 1996.

[7] C. T. Chen and W. D. Chang, "A feedforward neural network with function shape autotuning," *Neural Networks*, vol. 9, no. 4, pp. 627–641, 1996.

[8] T. Chen and H. Chen, "Approximations of continuous functionals by neural networks with application to dynamic systems," *IEEE Trans. Neural Networks*, vol. 4, pp. 910–918, Nov. 1993.

[9] ——, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[10] M. Craven and J. Shavlik, "Understanding time series networks: A case study in rule extraction," *Int. J. Neural Syst.*, vol. 8, no. 4, pp. 373–384, 1997.

[11] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Math. Contr., Signals, Syst.*, vol. 2, pp. 303–314, 1989.

[12] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 2151–2157, 1991.

[13] ——, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

[14] S. Hu and P. Yan, "Level-by-level learning for artificial neural groups," *ACTA Electronica SINICA*, vol. 20, no. 10, pp. 39–43, 1992.

[15] Z. Hu and H. Shao, "The study of neural network adaptive control systems," *Contr. Decision*, vol. 7, pp. 361–366, 1992.

[16] N. Karayiannis and A. Venetsanopoulos, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation and Applications*. Boston, MA: Kluwer, 1993, ch. 7.

[17] M. Leshno, V. Lin, A. Pinkus, and S. Schoken, "Multilayer feedforward networks with a nonpolynomial activation can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, 1993.

[18] H. Lu, R. Setiono, and H. Liu, "NeuroRule: A connectionist approach to data mining," in *Proc. Very Large Databases VLDB'95*, San Francisco, CA, 1995, pp. 478–489.

[19] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, pp. 246–257, 1991.

[20] ——, "Approximation and radial-basis-function networks," *Neural Comput.*, vol. 5, pp. 305–316, 1993.

[21] N. Redding, A. Kowalczyk, and T. Downs, "Constructive high-order network algorithm that is polynomial time," *Neural Networks*, vol. 6, pp. 997–1010, 1993.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Computing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, ch. 8.

[23] R. Trippi and E. Turbon, Eds., *Neural Networks in Finance and Investing*. Chicago, IL: Irwin Professional Publishing, 1996.

[24] L. Vecci, F. Piazza, and A. Uncini, "Learning and approximation capabilities of adaptive spline activation function neural networks," *Neural Networks*, vol. 11, pp. 259–270, 1998.

[25] V. Vemuri and R. Rogers, *Artificial Neural Networks: Forecasting Time Series*. Piscataway, NJ: IEEE Comput. Soc. Press, 1994.

[26] S. Xu and M. Zhang, "MASFinance, a model auto-selection financial data simulation software using NANNs," in Proc. Int. Joint Conf. Neural Networks—IJCNN'99, Washington, DC, 1999.

[27] ——, "Adaptive higher order neural networks," in *Proc. Int. Joint Conf. Neural Networks—IJCNN'99*, Washington, DC, 1999. (Paper #69).

[28] ——, "Approximation to continuous functionals and operators using adaptive higher order neural networks," in *Proc. Intl. Joint Conf. Neural Networks—IJCNN'99*, Washington, DC, 1999.
[29] T. Yamada and T. Yabuta, "Remarks on a neural network controller which uses an auto-tuning method for nonlinear functions," in *Proc. Joint Conf. Neural Networks*, vol. 2, New York, 1992, pp. 775–780.
[30] J. Zhang, M. Zhang, and J. Fulcher, "Financial prediction using higher order trigonometric polynomial neural network group models," *Proc. IEEE Int. Conf. Neural Networks*, pp. 2231–2234, June 8–12, 1997.
[31] ——, "Financial simulation system using higher order trigonometric polynomial neural network models," *Proc. IEEE/IAFE Conf. Comput. Intell. Financial Eng.*, pp. 189–194, Mar. 23–25, 1997.
[32] M. Zhang, S. Murugesan, and M. Sadeghi, "Polynomial higher order neural network for economic data simulation," in *Proc. Int. Conf. Neural Inform. Processing*, Beijing, China, 1995, pp. 493–496.
[33] M. Zhang and J. Fulcher, "Neural network group models for financial data simulation," in *Proc. World Congr. Neural Networks*, San Diego, CA, 1996, pp. 910–913.
[34] M. Zhang, J. Fulcher, and R. Scofield, "Rainfall estimation using artificial neural-network groups," *Neurocomput.*, vol. 16, no. 2, pp. 97–115, 1997.
[35] M. Zhang, S. Xu, and B. Lu, "Neuron-adaptive higher order neura-network group models," in *Proc. Intl. Joint Conf. Neural Networks—IJCNN'99*, Washington, DC, 1999. (Paper #71).
[36] M. Zhang, S. Xu, N. Bond, and K. Stevens, "Neuron-adaptive feedforward neural network group models," in *Proc. IASTED Int. Conf. Artificial Intell. Soft Comput.*, Honolulu, HI, Aug. 9–12, 1999, pp. 281–284.
[37] M. Zhang, J. Zhang, and J. Fulcher, "Higher order neural-network group models for financial simulation," *Int. J. Neural Syst.*, vol. 12, no. 2, pp. 123–142, 2000.

**Ming Zhang** (A'93–M'94–SM'95) was born in Shanghai, China. He received the M.S. degree in information processing and the Ph.D. degree in computer vision from East China Normal University, Shanghai, China, in 1982 and 1989, respectively.

He held Postdoctoral Fellowships in artificial neural networks with first the Chinese Academy of the Sciences in 1989, and then the United States National Research Council in 1991. He was also the recipient of a Senior Research Associate Fellowship from the latter organization in 1999. He is currently an Associate Professor in Computer Science at Christopher Newport University, Newport News, VA. He has more than 100 published papers, and his current research interests include artificial neural network models for management, financial data simulation, weather forecasting, and face recognition.

**Shuxiang Xu** received the B.Sc. degree in mathematics and the M.Sc. in applied mathematics from the University of Electronic Science and Technology of China, Chengdu, in 1989 and 1996, respectively. He received the Ph.D. degree in computing from the University of Western Sydney, Sydney, Australia, in 2000.

His current research interests include the theory and application of artificial neural networks (ANNs), especially the application of ANNs in financial simulation and forecasting, and in image recognition. He is currently a Lecturer with the School of Computing, University of Tasmania, Australia.

Dr. Xu was the recipient of an Australian Overseas Postgraduate Research Award.

**John Fulcher** (M'79–SM'99) is currently a Professor in the School of Information Technology and Computer Science, University of Wollongong, Wollongong, Australia. He has more than 70 publications, including a best-selling textbook on Microcomputer Interfacing, as well as three chapters in the *Handbook of Neural Computation* (Oxford, U.K.: Oxford University Press, 1997). He serves as a reviewer for several journals, especially in neural networks and computer education. He also serves on the Editorial Board of *Computer Science Education* (CSE) and has been Guest Editor for special issues of CSE (Australasia) and Computer Standards and Interfaces (Artificial Neural Networks).

Dr. Fulcher is a member of the Institute for Mathematical Modeling and Computational Systems at the University of Wollongong and ACM.