Artificial Neural Networks in Finance and Manufacturing

Joarder Kamruzzaman
Monash University, Australia

Rezaul K. Begg
Victoria University, Australia

Ruhul A. Sarker
University of New South Wales, Australia

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore

Acquisitions Editor:	Michelle Potter
Development Editor:	Kristin Roth
Senior Managing Editor:	Amanda Appicello
Managing Editor:	Jennifer Neidig
Copy Editor:	Chuck Pizar
Typesetter:	Cindy Consonery
Cover Design:	Lisa Tosheff
Printed at:	Yurchak Printing Inc.

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanonline.com

Library of Congress Cataloging-in-Publication Data

Artificial neural networks in finance and manufacturing / Joarder Kamruzzaman, Rezaul Begg and Ruhul Sarker, editors.
p. cm.
Summary: "This book presents a variety of practical applications of neural networks in two important domains of economic activity: finance and manufacturing"--Provided by publisher.
Includes bibliographical references and index.
ISBN 1-59140-670-6 (hardcover) -- ISBN 1-59140-671-4 (softcover) -- ISBN 1-59140-672-2 (ebook)
1. Neural networks (Computer science)--Economic aspects. 2. Neural networks (Computer science)--Industrial applications. 3. Finance--Computer simulation. 4. Manufacturing processes--Computer simulation. I. Kamruzzaman, Joarder. II. Begg, Rezaul. III. Sarker, Ruhul A.
HD30.2.N48 2006
332.0285'632--dc22
2006003560

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter V

Application of Higher-Order Neural Networks to Financial Time-Series Prediction

John Fulcher, University of Wollongong, Australia

Ming Zhang, Christopher Newport University, USA

Shuxiang Xu, University of Tasmania, Australia

Abstract

Financial time-series data is characterized by nonlinearities, discontinuities, and high-frequency multipolynomial components. Not surprisingly, conventional artificial neural networks (ANNs) have difficulty in modeling such complex data. A more appropriate approach is to apply higher-order ANNs, which are capable of extracting higher-order polynomial coefficients in the data. Moreover, since there is a one-to-one correspondence between network weights and polynomial coefficients, higher-order neural networks (HONNs) — unlike ANNs generally — can be considered open-, rather than "closed-box" solutions, and thus hold more appeal to the financial community. After developing polynomial and trigonometric HONNs (P[T]HONNs), we introduce the concept of HONN groups. The latter incorporate piecewise continuous-activation *functions and thresholds, and as a result are capable of modeling discontinuous (or piecewise-continuous) data, and what is more to any degree of accuracy. Several other PHONN variants are also described. The performance of P(T)HONN and HONN groups on representative financial time series is described (i.e., credit ratings and exchange rates). In short, HONNs offer roughly twice the performance of MLP/BP on financial time-series prediction, and HONN groups around 10% further improvement.*

Financial Time Series Prediction

It is clear that there are pattern(s) underlying some time series. For example, the 11-year cycle observed in sunspot data (University of California, Irvine, 2005). Whether this is the case with financial time-series data is debatable. For instance, do underlying "forces" actually drive financial markets, and if so can their existence be deduced by observations of stock price and volume movements (Back, 2004)?

Alternatively, do so-called "market inefficiencies" exist, whereby it is possible to devise strategies to consistently "beat the market" in terms of return-on-investment (Edelman & Davy, 2004)? If this is in fact the case, then it runs counter to the so-called Efficient Markets Hypothesis, namely that the present pricing of a financial asset is a reflection of all the available information about that asset, whether this be private (insider), public, or previous pricing (if based *solely* on the latter, then this is referred to as the "weak form" of the EMH).

Market traders, by contrast, tend to base their decisions not only on the previous considerations, but also on many other factors, including hunches (intuition). Quantifying these often complex decision-making processes (expertise) is a difficult, if not impossible, task akin to the fundamental problem inherent in designing *any* Expert System. An overriding consideration is that any model (system) tends to break down in the face of singularities, such as stock market crashes (e.g., "Black Tuesday", October 1987), war, political upheaval, business scandals, rumor, panic buying, and so on.

"Steady-state" markets, on the other hand, tend to exhibit *some* predictability, albeit minor — for example, so-called "calendar effects": lower returns on Mondays, higher returns on the last day of the month and just prior to public holidays, higher returns in January, and so on (Kingdon, 1997).

Now, while it is possible that financial time-series data on occasion can be described by a linear function, most often it is characterized by nonlinearities, discontinuities, and high-frequency multipolynomial components.

If there *is* an underlying market model, then it has remained largely impervious to statistical (and other forms of) modeling. We can take a lead here from adaptive control systems and/or machine learning; in other words, if a system is too complex to model, try *learning* it. This is where techniques such as ANNs can play a role.

Many different techniques have been applied to financial time-series forecasting over the years, ranging from conventional, model-based, statistical approaches to more esoteric, data-driven, experimental ones (Harris & Sollis, 2003; Mills, 1993; Reinsel, 1997).

Some examples of the former are Auto Regression (AR), ARCH, Box-Jenkins (Box & Jenkins, 1976), and Kalman Filter (Harvey, 1989). Some examples of the latter are ANNs (Zhang, Patuwo, & Hu, 1998), Fuzzy Logic and variants (Sisman-Yilmaz, Alpaslan, & Jain, 2004), Evolutionary Algorithms (Allen & Karjalainen, 1999; Chen, 2002), Genetic Programming (Chen, 2002; Iba & Sasaki, 1999), Support Vector Machines (Edelman & Davy, 2004; Tay & Cao, 2001), Independent Component Analysis (Back, 2004), and other so-called (often biologically inspired) “soft computing” techniques (Kingdon, 1997). We focus on ANNs in this chapter, more specifically on *higher-order* neural networks, for reasons that we shall elaborate upon shortly.

Artificial Neural Networks (ANNs)

When people speak of ANNs, they are most likely referring to feed-forward Multilayer Perceptrons (MLPs), which employ the backpropagation (BP) training algorithm (e.g., Lapedes & Farber, 1987; Refenes, 1994; Schoneberg, 1990). Following the lead of the M-competition for different forecasting techniques (Makridakis, Andersoen, Carbone, Fildes, Hibon, Lewandowski, et al., 1982), in which such ANNs compared favorably with the Box-Jenkins method, Weigand and Gershenfeld (1993) compared nonlinear forecasting techniques on a number of different time series, one of which being currency exchange rate. ANNs, along with state-space reconstruction techniques, fared well in this more recent comparative study.

At first sight, it would appear that MLP/BPs should perform reasonably well at financial time-series forecasting, since they are known to excel at (static) pattern recognition and/or classification; in this particular case, the patterns of interest are simply different time-shifted samples taken from the same data series.

Now Hornik (1991) has shown that an MLP with an arbitrary bounded nonconstant activation is capable of universal approximation. More specifically, a single hidden layer MLP/BP can approximate arbitrarily closely any suitably smooth function (Hecht-Nielsen, 1987; Hornik, Stinchcombe, & White, 1989). Furthermore, this approximation improves as the number of nodes in the hidden layer increases. In other words, a suitable network can always be found.

A similar but more extended result for learning conditional probability distributions was found by Allen and Taylor (1994). Here, two network layers are required in order to produce a smooth limit when the stochastic series (such as financial data) being modeled becomes noise free.

During learning, the outputs of a supervised neural network come to approximate the target values given the inputs in the training set. This ability may be useful in itself, but more often the purpose of using a neural net is to generalize — in other words, to have the network outputs approximate target values given inputs that are *not* in the training set.

Generally speaking, there are three conditions that are typically necessary — although not sufficient — for good generalization.

The first necessary condition is that the network inputs contain sufficient information pertaining to the target, so that there exists a mathematical function relating correct outputs to inputs with the desired degree of accuracy (Caudill & Butler, 1990).

The second necessary condition is that the function we are attempting to learn (relating inputs to desired outputs) be, in some sense, smooth (Devroye, Gyorfi, & Lugosi, 1996; Plotkin, 1993). In other words, small changes in inputs should produce small changes in outputs, at least most of the time. For continuous inputs and targets, function smoothness implies continuity and restrictions on the first derivative over most of the input space. Now some neural networks — including the present authors' HONN models — are able to learn discontinuities, provided the function consists of a finite number of continuous pieces. Conversely, very nonsmooth functions (such as those produced by pseudorandom number generators and encryption algorithms) are not able to be generalized by standard neural networks.

The third necessary condition for good generalization is that the training exemplars constitute a sufficiently large and representative subset ("sample" in statistics terminology) of the set of all cases we want to generalize to (the "population" in statistics terminology) (Wolpert, 1996a, 1996b). The importance of this condition is related to the fact that there are, generally speaking, two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are more or less surrounded by nearby training cases; everything else is extrapolation. In particular, cases that are outside the range of the training data require extrapolation. Cases inside large "holes" in the training data may also effectively require extrapolation. Interpolation can often be performed reliably, but extrapolation is notoriously unreliable. Hence, it is important to have sufficient training data to avoid the need for extrapolation. Methods for selecting good training sets are discussed in numerous statistical textbooks on sample surveys and experimental design (e.g., Diamond & Jeffries, 2001).

Despite the universal approximation capability of MLP/BP networks, their performance is limited when applied to financial time-series modeling and/or prediction (forecasting). This is due in part to two limitations of feed-forward ANNs, namely (Zhang, Xu, & Fulcher, 2002):

1. Their activation functions have fixed parameters only (e.g., sigmoid, radial-basis function, and so on), and
2. They are capable of continuous function approximation only; MLPs are unable to handle discontinuous and/or piecewise-continuous (economic) time-series data.

Networks with *adaptive* activation functions seem to provide better fitting properties than classical architectures with *fixed* activation-function neurons. Vecci, Piazza, and Uncini (1998) studied the properties of a feed-forward neural network (FNN) which was able to adapt its activation function by varying the control points of a Catmull-Rom cubic spline. Their simulations confirmed that the special learning mechanism allows us to use the network's free parameters in a very effective way. In Chen and Chang (1996), real variables *a* (gain) and *b* (slope) in the generalized sigmoid activation function were adjusted during the learning process. They showed that from the perspective of static and dynamical system modeling, use of adaptive sigmoids (in other words, sigmoids with free parameters) leads to improved data modeling compared with classical FNNs. Campolucci, Capparelli, Guarnieri, Piazza, and Uncini (1996) built an adaptive activation function as a piecewise approximation with suitable cubic splines. This function had arbitrary shape and allowed the overall size of the neural network to be reduced, trading connection complexity against activation function complexity. Several other authors (Hu & Shao, 1992; Yamada & Yabuta, 1992) have also studied the properties of neural networks that utilize adaptive activation functions.

In short, some researchers have devoted their attention to more sophisticated, alternative ANN models. One natural extension is to incorporate unit time delays (memory elements) to turn the MLP/BP into a recurrent network, in order to recognize (classify) dynamic rather than static input patterns. Alternatively, replication of network nodes and weights across time leads to time-delay neural networks, in which the layer inputs are time-shifted versions from the same time-series data. Such attempts to incorporate temporal units into an ANN have not usually led to significant improvements in financial time-series modeling/predicting performance though.

Higher-Order Neural Networks (HONNs)

Traditional areas in which ANNs are known to excel are pattern recognition, pattern matching, and mathematical function approximation (nonlinear regression). However, they suffer from several well-known limitations. They can often become stuck in local, rather than global minima, as well as taking unacceptably long times to converge in practice. Of particular concern, especially from the perspective of financial time-series prediction, is their inability to handle nonsmooth, discontinuous training data and complex mappings (associations). Another limitation of ANNs is their “black box” nature — meaning that explanations (reasons) for their decisions are not immediately obvious, unlike some other techniques, such as decision trees.

This then is the motivation for developing higher-order neural networks (HONNs).

Background on HONNs

The term “higher-order” neural network can mean different things to different people, ranging from a description of the neuron activation function to preprocessing of the neuron inputs, signifying connections to more than one layer or just ANN functionality (in other words, their ability to extract higher-order correlations from the training data). In this chapter, we use “HONN” to refer to the incorporation of a range of neuron types: linear, power, multiplicative, sigmoid, and logarithmic (see Figure 3).

HONNs have traditionally been characterized as those in which the input to a computational neuron is a weighted sum of the products of its inputs (Lee et al., 1986). Such neurons are sometimes called higher-order processing units (HPUs) (Lippmann, 1989). It has been established that HONNs can successfully perform invariant pattern recognition (Psaltis, Park, & Hong, 1988; Reid, Spirkovska, & Ochoa, 1989; Wood & Shawe-

Taylor, 1996). Giles and Maxwell (1987) showed that HONNs have impressive computational, storage, and learning capabilities. Redding, Kowalski and Downs (1993) proved that HONNs were at least as powerful as any other (similar order) FNN. Kosmatopoulos, Polycarpou, Christodoulou, & Ioannou (1995) studied the approximation and learning properties of one class of recurrent HONNs and applied these architectures to the identification of dynamical systems. Thimm and Fiesler (1997) proposed a suitable initialization method for HONNs and compared this with FNN-weight initialization.

First-order neural networks can be formulated as follows, assuming simple McCullough-and-Pitts-type neurons (Giles & Maxwell, 1987):

$$y_i(x) = f\left[\sum_{j}^{N} W(i,j)x(j)\right] \tag{1}$$

where **{x(j)}** = an N-element input vector, W(i,j) = adaptable weights from all other neurons to neuron-i, and *f* = neuron threshold function (e.g., sigmoid). Such neurons are said to be linear, since they are only capable of capturing first-order correlations in the training data. In this sense, they can be likened to Least Mean Squared or Delta learning, as used in ADALINE. It is well known that Rosenblatt's original (two-layer) perceptron was only capable of classifying linearly separable training data. It was not until the emergence of *Multi*layer Perceptrons (which incorporated nonlinear activation functions, such as sigmoid) that more complex (*non*linear) data could be discriminated.

Higher-order correlations in the training data require more complex neuron activation functions, characterized as follows (Barron, Gilstrap, & Shrier, 1987; Giles & Maxwell, 1987; Psaltis, Park, & Hong, 1988):

$$y_i(x) = f\left[W_0(i)\sum_{j}^{N} W_i(i,j)x(j) + \sum_{j}^{N}\sum_{k}^{M} W_i(i,j,k)x(j)x(k) + \ldots\right] \tag{2}$$

Figure 1. Higher-order neural network architecture-I

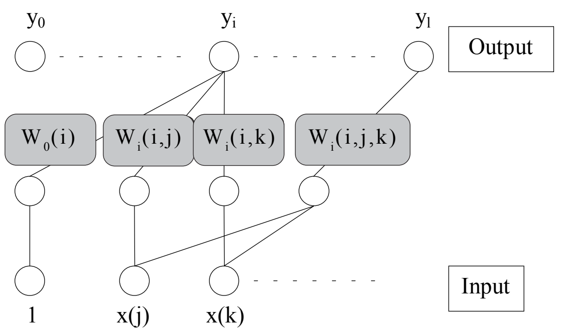
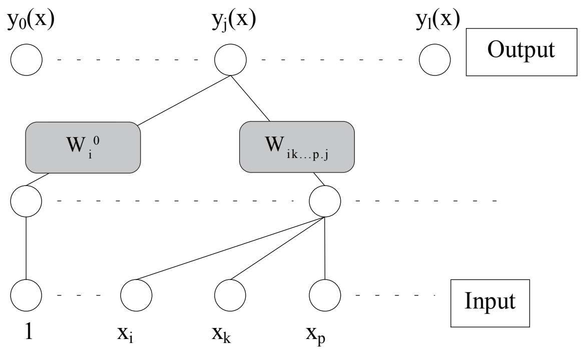


Figure 2. Higher-order neural network architecture-II



Neurons that include terms up to and including degree-k are referred to as k*th*-order neurons (nodes). Figure 1 further explains the subscripts *i*, *j*, and *k* used in Equation 2.

The following alternative, simpler formulation is due to Lisboa and Perantonis (1991):

$$y_j(x) = f\left[W_i^0 + \sum_i \sum_k \ldots \sum_p w_{ik\ldots p,j} x_i, x_k \ldots x_p\right] \tag{3}$$

where a single weight is applied to all n-tuples $x_i \ldots x_p$ in order to generate output-y_i from that particular neuron.

This is reminiscent of Rumelhart, Hinton, and Williams (1986) formulation of their so-called "sigma-pi" neurons ($\sum w_{ij} \prod xi_1 xi_2 \ldots xi_k$), for which they show that the generalized Delta Rule (standard backpropagation) can be applied as readily as for simple additive neurons ($\sum w_{ij} x_i$). Moreover, the increased computational load resulting from the large increase in network weights means that the complex input-output mappings, normally only achievable in multilayered networks, can now be realized in a *single* HONN layer (Zhang & Fulcher, 2004).

In summary, HONN activation functions incorporate *multiplicative* terms.

Now the output of a k*th*-order single-layer HONN neuron will be a nonlinear function comprising polynomials of up to k*th*-order. Moreover, since no hidden layers are involved, both Hebbian and perceptron learning rules can be employed (Shin & Ghosh, 1991).

Multiplicative interconnections within ANNs have been applied to many different problems, including invariant pattern recognition (Giles, Griffin, & Maxwell, 1988; 1991; Goggin, Johnson, & Gustafson, 1993; Lisboa & Pentonis, 1991), however their complexity usually limits their usefulness.

Karayiannis and Venetsanopoulos (1993) make the observation that the performance of first-order ANNs can be improved, within bounds, by utilizing sophisticated learning algorithms. By contrast, HONNs can achieve superior performance *even if* the learning algorithm is based on the simpler outer-product rule.

A different approach was taken by Redding, Kowalczy, and Downs (1993) and involved the development of a constructive HONN architecture that solved the binary mapping in polynomial time. Central to this process was the selection of the multiplicative nonlinearities as hidden nodes within the HONN, depending on their relevance to the pattern data of interest.

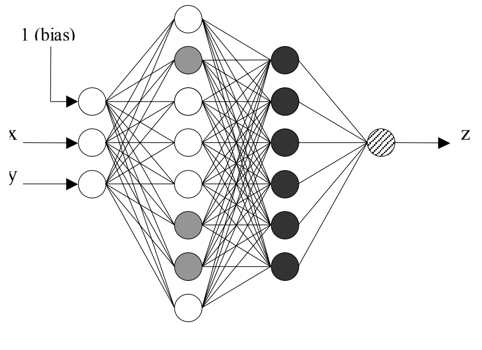
Polynomial HONNs

The authors have developed several different HONN models during the past decade or so. We now present a brief background on the development of polynomial, trigonometric, and similar HONN models. A more comprehensive coverage, including derivations of weight-update equations, is presented in Zhang and Fulcher (2004).

Firstly, all PHONNs described in this section utilize various combinations of linear, power, and multiplicative (and sometimes other) neuron types and are trained using standard backpropagation. The generic HONN architecture is shown in Figure 3, where there are two network inputs (independent variables) x and y, and a single network output (dependent variable) z.

In the first hidden layer, the white neurons are either $\cos(x)$ or $\sin(y)$, and the grey neurons either $\cos^2(x)$ or $\sin^2(y)$. All (black) neurons in the second hidden layer are multiplicative, and the (hashed) output neurons either linear (PHONN#1) or a sigmoid-logarithmic pair (PHONN#2), as described later. Some of the intermediate weights are fixed and some variable, according to the formulation of the polynomial being synthesized. All of the weights connecting the second hidden layer to the output layer are adaptable (PHONN#1,2). By contrast, only the latter are adjustable in PHONN#0; the first two-layer weights are fixed (=1).

Figure 3. Polynomial higher-order neural network (PHONN)



The first model (PHONN#0) facilitates extraction of the linear coefficients a_{k1k2} from the general n*th*-order polynomial:

$$z_i(x,y) = \sum_{k1k2} a_{k1k2} x_i^{k1} y_i^{k2} \tag{4}$$

Now, since variable weights are present in only one layer here, PHONN#0 can be compared with Rosenblatt's (two-layer) perceptron, which is well-known to be limited to solving linearly separable problems.

In PHONN#1, the general *n*th-order polynomial of Equation 4 is expanded as follows:

$$z_i(x,y) = \sum_{k1k2=0}^{n} (a_{k1k2}^{0})[a_{k1k2}^{x} x]^{k1} [a_{k1k2}^{y} y]^{k2} \tag{5}$$

Each coefficient from Equation 4 has now been replaced by three terms in Equation 5. Moreover, we now have *two* adjustable layers at our disposal, such that PHONN#1 has similar discrimination capability to an MLP.

The linear output neuron of PHONN#1 is replaced by a sigmoid-logarithmic neuron pair in PHONN#2, which leads to faster network convergence. Model PHONN#3 comprises groups of PHONN#2 neurons (ANN groups are discussed in the following section).

If we use a PHONN to simulate the training data, the model will "learn" the coefficients and order of the polynomial function. If we use adaptive HONN models to simulate the data, the models will *not only* "learn" the coefficients and order, *but also* the different

Table 1. $A-$US exchange rate (March 2005)

Date	Exchange Rate	Input#1 (X)	Input#2 (Y)	Desired Output (Z)
1	0.7847	0.093	0.000	0.057
2	0.7834	0.000	0.057	0.607
3	0.7842	0.057	0.607	0.650
4	0.7919	0.607	0.650	1.000
7	0.7925	0.650	1.000	0.686
8	0.7974	1.000	0.686	0.479
9	0.7930	0.686	0.479	0.729
10	0.7901	0.479	0.729	0.229
11	0.7936	0.729	0.229	0.429
14	0.7866	0.229	0.429	0.800
15	0.7894	0.429	0.800	0.714
16	0.7946	0.800	0.714	0.736
17	0.7935	0.714	0.736	
18	0.7937	0.736		

functions. In other words, the model "learns" the polynomial function if the data is in fact a polynomial function.

Now, instead of employing a polynomial series expansion, we could alternatively use a trigonometric one, as indicated in Equation 6.

$$Z = a_{00} + a_{01}\sin(y)\ a_{02}\sin^2(y) + a_{10}\cos(x) + a_{11}\cos(x)\sin(y) + a_{12}\cos(x)\sin^2(y) + a_{20}\cos^2(y) + a_{21}\cos^2(x)\sin(y) + a_{22}\cos^2(x)\sin^2(y) + \ldots \tag{6}$$

This naturally leads to the development of THONN models (and likewise THONN groups — see subsequent paragraphs).

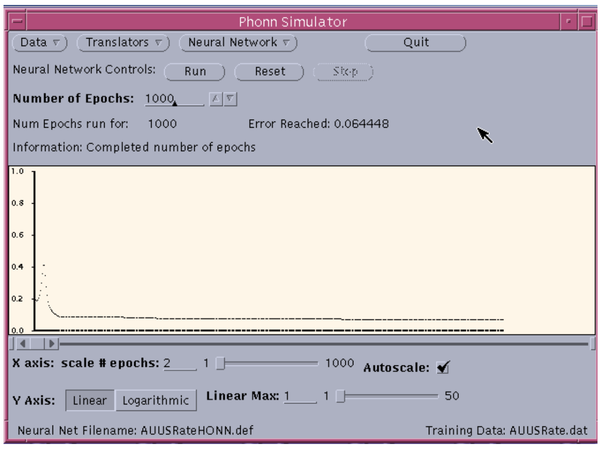
One significant feature of models P(T)HONN#1 and higher is that we have opened up the "black box" or closed architecture normally associated with ANNs. In other words, we are able to associate individual network weights with polynomial coefficients and vice versa. This is a significant finding, since users — especially those in the financial sector — invariably prefer explanations (justifications) for the decisions made by their predictors, regardless of the nature of the underlying decision engine.

We now proceed to illustrate the use of PHONNs by way of a simple example, namely exchange-rate prediction. Table 1 shows how the Australian-US dollar exchange rate varied during March 2005 (Federal Reserve Board, 2005).

The following formula was used to scale the data to within the range 0 to 1, in order to meet constraints:

$$\{(individual\_rate) - (lowest\_rate)\} \Big/ \{(highest\_rate) - (lowest\_rate)\} \tag{7}$$

Figure 4. PHONN Simulator main window



Neural Network Controls:
Number of Epochs: 1000
Num Epochs run for: 1000 Error Reached: 0.064448
Information: Completed number of epochs
X axis: scale # epochs: 2 1 1000 Autoscale: ✓
Y Axis: [✓ Linear] Logarithmic Linear Max: 1 1 50
Neural Net Filename: AUUSRateHONN.def Training Data: AUUSRate.dat

Equation 7 was applied to each separate entry of a given set of simulation data — in other words, the *individual_rate*. The smallest entry in the data set serves as the *lowest_rate*, and the largest entry ss the *highest_rate*. After applying Equation 7, the data have been converted into the Input#1 column of Table 1.

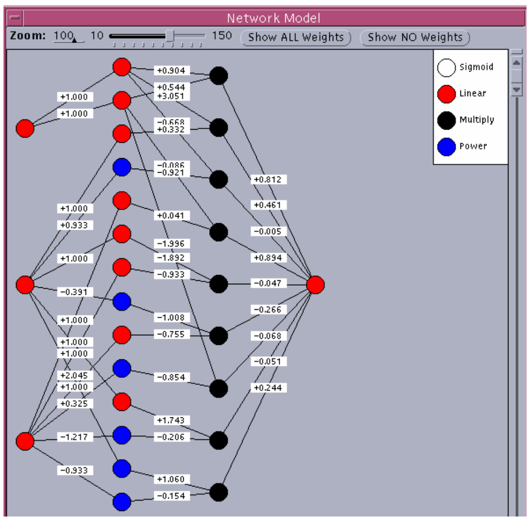
We use previous day and current-day exchange rates to predict the next day's rate. Accordingly, we copy the data from 3/2/2005 to 3/18/2005 to the Input#2 column of Table 1 — this being the second input to the PHONN — and copy the data from 3/3/2005 to 3/18/2005 to the Output column of Table 1 (in other words, the desired PHONN output).

The PHONN simulation system was written in the C language, and runs under X-Windows on a SUN workstation. It incorporates a user-friendly graphical user interface (GUI), which enables any step, data, or calculation to be reviewed and modified dynamically in different windows. At the top of the PHONN simulator main window are three pull-down menus: Data, Translators, and Neural Network, as illustrated in Figure 4 (which, by the way, shows the result of network training using the data of Table 1).

Each of these offers several options, and selecting a particular option creates another window for further processing. For instance, once we have selected a data set via the Data menu, two options are presented for data loading and graphical display.

Data is automatically loaded when the Load option is selected. Alternatively, the Display option displays data not only in graphical form, but also translated, if so desired (e.g., rotation, elevation, grids, smooth, influence, etc.). The Translators menu is used to convert the selected raw data into network form, while the Neural Network menu is used to convert the data into a nominated model (an example of which appears in Figure 5). These two menus allow the user to select different models and data, in order to generate

Figure 5. "PHONN Network Model" subwindow



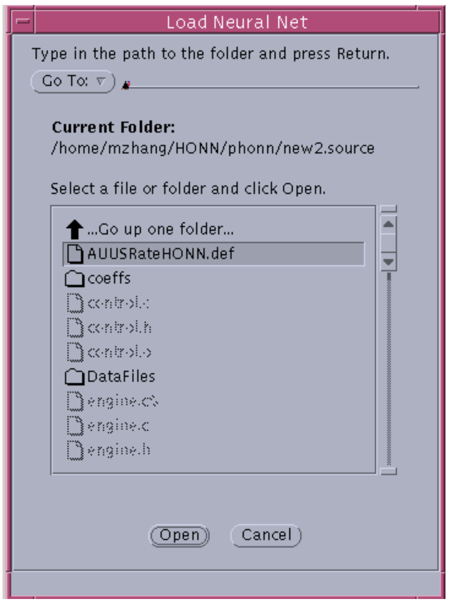
and compare results. Figure 5 shows the network weights resulting from training using the data of Table 1.

All of the previous steps can be simply performed using a mouse. Hence, changing data or network model and comparing results can all be achieved easily and efficiently.

There are more than twelve windows and subwindows in the PHONN Simulator system; both the system mode and its operation can be viewed dynamically, in terms of:

- Input/output data,
- Neural network models,
- Coefficients/parameters, and so on.

Figure 6. "Load Network Model File" subwindow



Load Neural Net

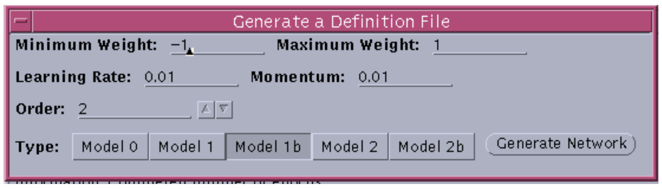
Type in the path to the folder and press Return.

Current Folder:
/home/mzhang/HONN/phonn/new2.source

Select a file or folder and click Open.

...Go up one folder...
AUUSRateHONN.def
coeffs
control.c
control.h
control.o
DataFiles
engine.c%
engine.c
engine.h

Figure 7. "Generate Definition File" subwindow



Generate a Definition File

Minimum Weight: -1 **Maximum Weight:** 1

Learning Rate: 0.01 **Momentum:** 0.01

Order: 2

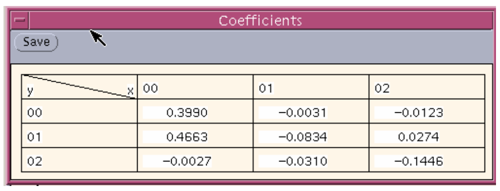
Type: Model 0 | Model 1 | Model 1b | Model 2 | Model 2b

The simulator operates as a general neural network system and includes the following functions:

- Load a data file,
- Load a neural network model (Figure 6) for Table 1 data,
- Generate a definition file (Figure 7) for Table 1 data,
- Write a definition file,
- Save report,
- Save coefficients (Figure 8) for Table 1 data, and so on.

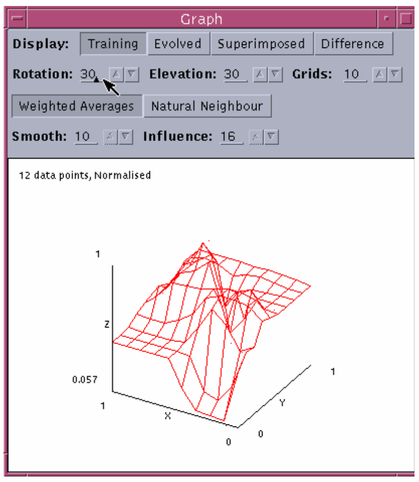
The "System mode" windows allow the user to view, in real time, how the neural network model learns from the input training data (in other words, how it extracts the weight values).

Figure 8. "Coefficients" subwindow



y \ x	00	01	02
00	0.3990	−0.0031	−0.0123
01	0.4663	−0.0834	0.0274
02	−0.0027	−0.0310	−0.1446

Figure 9. "Graph" subwindow



Rotation: 30 Elevation: 30 Grids: 10

Smooth: 10 Influence: 16

12 data points, Normalised

When the system is running, the following system-mode windows can be opened simultaneously from within the main window:

- "Display Data (Training, Evolved, Superimposed, and Difference),"
- "Show/Set Parameters,"
- "Network Model (including all weights)," and
- "Coefficients."

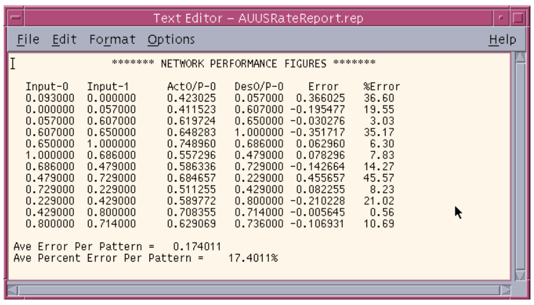
Thus, every aspect of the system's operation can be viewed graphically.

A particularly useful feature of this system is that one is able to view the mode, modify it, or alternatively change other parameters *in real time*. For example, when the user chooses the "Display Data" window to view the input-training data file, they can change the graph format for the most appropriate type of display (in other words, modify the graph's rotation, elevation, grids, smoothing, and influence).

During data processing, the "Display Data" window offers four different models to display the results, which can be changed in real time, namely: "Training," "Evolution," "Superimposed," and "Difference (using the same format selected for the input data)," as indicated in Figure 9 for Table 1 data.

- "Training" displays the data set used to train the network,
- "Evolved" displays the data set produced by the network (and is unavailable if a network definition file has not been loaded),
- "Superimposed" displays both the training and the evolved data sets together (so they can be directly compared in the one graph), and
- "Difference" displays the difference between the "Training" and the "Evolved" data sets.

Figure 10. Report generation within PHONN Simulator


Text Editor – AUUSRateReport.rep
File Edit Format Options Help

```
              ******* NETWORK PERFORMANCE FIGURES *******

  Input-0   Input-1     Act0/P-0   Des0/P-0    Error    %Error
  0.093000  0.000000    0.423025   0.057000  0.366025   36.60
  0.000000  0.057000    0.411523   0.607000 -0.195477   19.55
  0.057000  0.607000    0.619724   0.650000 -0.030276    3.03
  0.607000  0.650000    0.648283   1.000000 -0.351717   35.17
  0.650000  1.000000    0.748960   0.686000  0.062960    6.30
  1.000000  0.686000    0.557296   0.479000  0.078296    7.83
  0.686000  0.479000    0.586336   0.729000 -0.142664   14.27
  0.479000  0.729000    0.684657   0.229000  0.455657   45.57
  0.729000  0.229000    0.511255   0.429000  0.082255    8.23
  0.229000  0.429000    0.589772   0.800000 -0.210228   21.02
  0.429000  0.800000    0.708355   0.714000 -0.005645    0.56
  0.800000  0.714000    0.629069   0.736000 -0.106931   10.69

Ave Error Per Pattern =   0.174011
Ave Percent Error Per Pattern =    17.4011%
```

The "Rotation" command changes the angle of rotation at which the wire-frame mesh is projected onto the screen. This allows the user to "fly" around the wire-frame surface. The default value is 30º, but is adjustable from 0º to 355º in increments of 5º, with wraparound from 360º to 0º (this value can be simply adjusted with either the "up/down" buttons or by entering a number directly).

"Elevation" changes the angle of elevation at which the wire-frame mesh is projected onto the screen. This allows the user to "fly" either above or below the wire-frame surface (usage is similar to Rotation).

The "Grids" command changes the number of wires used in the wire-frame mesh. It is adjustable between 6 and 30, using either the "up/down" buttons or by directly entering a number. Low grid numbers allow fast display, but with decreased resolution; high numbers provide a more accurate rendition of the surface, but at the cost of increased display time.

If the user is not satisfied with the results and wants a better outcome (a higher degree of model accuracy), they can stop the processing and set *new* values for the model parameters, such as learning rate, momentum, error threshold, and random seed. The neural network model can be easily changed as well.

As usual with neural network software, the operating procedure is as follows:

Step 1: Data pre-processing (encoding),

Step 2: Load and view data,

Step 3: Choose and load neural network model,

Step 4: Show/Set the network parameters,

Step 5: Run the program,

Step 6: Check the results:

    If satisfactory, then go to Step 7, otherwise go to Step 3,

Step 7: Save and export the results,

Step 8: Data decoding (postprocessing).

There are two basic requirements that must be satisfied before the PHONN simulator is able to start running: One is *input training data* and the other is *input the network*. The users must also have *loaded some training data* and *loaded a network*.

Figure 10 shows the running report for Table 1 data, from which we see the average error is 17.4011%. Returning to Figure 8, we see that the following formula can be used to represent the data of interest (exchange rate), thereby relating network weights to polynomial coefficients:

$$Z=0.3990-0.0031X-0.0123X*X+0.4663Y-0.0834X*Y-0.0274X*X*Y \\ -0.0027Y*Y-0.0310X*Y*Y-0.1446X*X*Y*Y \qquad (8)$$

HONN Groups

Prior to our development of ANN groups, we were aware of earlier work on groups of individual neurons (Hu & Pan, 1992; Willcox, 1991). What motivated our development of firstly ANN groups and thenceforth P(T)HONN groups was the poor performance of ANNs on human-face recognition, which we investigated in the context of airport security (Zhang & Fulcher, 1996).

It is possible to define a neural network group in the usual set theory terms, as follows:

$$ANN = MLP \cup SOM \cup RBF \cup ART \cup SVM \ldots \tag{9}$$

MLP is thus a subset of the set *ANN*; likewise a particular instance of *MLP* (say *MLP*100:70:20) is a subset of *MLP*. Moreover, providing either the sum and/or product can be defined for every two elements in a nonempty set $N \subset ANN$ (Inui, Tanabe & Onodera, 1978; Naimark & Stern, 1982), and then we refer to this set as a neural network group.

ANN groups are particularly useful in situations involving discontinuous data, since we can define piecewise function groups, as follows:

$$\begin{aligned} O_i + O_j &= O_i && (A < I < B) \\ &= O_j && (B < I < C) \end{aligned} \tag{10}$$

where I = ANN input, O = ANN output, and for every two elements $n_i, n_j \ni N$, the sum $n_i + n_j$ is a piecewise function.

Now in the same vein as Hornik (1991) and Leshno (1993), it is possible to show that piecewise function groups (of MLPs employing locally bounded, piecewise continuous activation functions and thresholds) are capable of approximating any piecewise continuous function, to any degree of accuracy (Zhang, Fulcher, & Scofield, 1997).

Not surprisingly, such ANN groups offer superior performance compared with ANNs when dealing with discontinuous, nonsmooth, complex training data, which is often the case with financial time series.

HONN Applications

The three application areas in which we have focused our endeavors to date are (1) human-face recognition (Zhang & Fulcher, 1996), (2) satellite weather forecasting (Zhang, Fulcher, & Scofield, 1997; Zhang & Fulcher, 2004), and (3) financial time-series prediction (Zhang, Xu, & Fulcher, 2002; Zhang, Zhang, & Fulcher, 2000). In each case, we are typically dealing with discontinuous, nonsmooth, complex training data, and thus HONN (and HONN groups) come into their own.

Automatic Face Recognition

Automatic (1-of-*n*) facial recognition is a complex pattern recognition task, especially for real-time operation under variable lighting conditions, where faces are tilted or rotated, and where *n* is large. There can also be significant repercussions for false positives, and more especially false negatives (that is, failure to detect a "wanted person"). The advantage of using ANN groups in such an application is that if one particular ANN model cannot perform the desired recognition, then perhaps another model belonging to the *ANN* set (group) can do better.

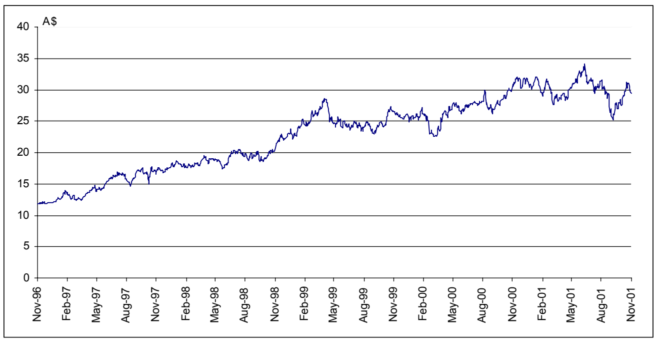
Using ANN group trees can extend this approach further. Nodes and interconnecting weights in such trees grow adaptively during the training process, according to both the desired number of "wanted" leaf-node faces and the variability contained within the training exemplars. As a result, such a network is capable of recognizing tilted or rotated facial images as being the same person; in other words, it can handle topological deformations and/or 3D translations. Zhang and Fulcher (1996) describe such ANN group trees in terms of Tolerance Space Theory (Chen, 1981; Zeeman, 1962).

Group-based adaptive-tolerance (GAT) trees have been successfully applied to automatic face recognition (Zhang & Fulcher, 1996). For this study, ten (28*28 pixel, 256-level gray scale) images of 78 different faces (front, tilted, rotated, smiling, glasses, beard, etc.) were used for both training (87) and testing (693) purposes. For front-face recognition, the error rate was 0.15% (1 face); for tilted and rotated faces (of up to15%), the error rates were 0.16% and 0.31%, respectively. Thus GAT trees were more "tolerant" in their classification.

Rainfall Estimation

Global weather prediction is acknowledged as one of computing's "grand challenges" (Computing Research Associates, 2005). The world's fastest (parallel vector)

Figure 11. Commonwealth Bank of Australia share prices (November 1996-November 2001)



supercomputer — at least up until 2004 (Meuer, Stronmaier, Dongarra, & Simon, 2005) — was devoted to simulation of the earth for purposes of global weather forecasting.

Rainfall estimation is a complicated, nonlinear, discontinuous process. Single ANNs are unable to deal with discontinuous, nonsmooth input training data; ANN groups, on the other hand, are well-suited to such problems.

ANNs and ANN groups both outperform conventional rainfall estimation, yielding error rates of around 17% and 3.9%, respectively (compared with ~30.4% with the latter) (Zhang & Fulcher, 2004; Zhang, Fulcher, & Scofield, 1997). ANN groups were subsequently used as the reasoning engine within the ANSER Expert System developed for satellite-derived rainfall estimation.

In Zhang and Fulcher (2004), PHONN variants (PT-, A- and M-) are applied to half-hourly rainfall prediction. Another model — the Neuron-Adaptive HONN (described in the next section) — led to a marginal error reduction (3.75%).

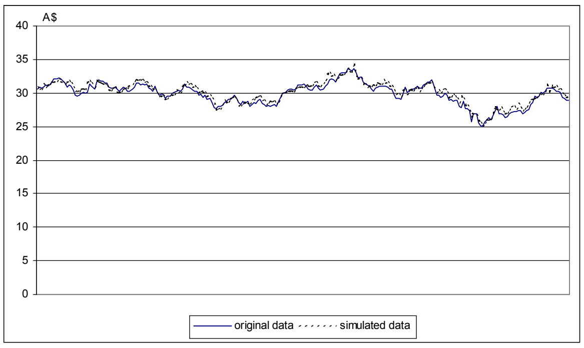
Another variant — the Sigmoid PHONN — has been shown to offer marginal performance improvement over both PHONN and M-PHONN when applied to rainfall estimation (more specifically, 5.263% average error compared with 6.36% and 5.42%, respectively) (Zhang, Crane, & Bailey, 2003).

Application of HONNs to Financial Time Series Data

Both polynomial and trigonometric HONNs have been used to both simulate and predict financial time-series data (Reserve Bank of Australia Bulletin, 2005), to around 90% accuracy (Zhang, Murugesan, & Sadeghi, 1995; Zhang, Zhang, & Keen, 1999).

In the former study, all the available data was used during training. In the latter, the data was split in two — one half being used for training and the other half for testing (and where data from the previous 2 months was used to predict the next month's data). More

Figure 12. Simulation of Commonwealth Bank of Australia share prices (November 2000-November 2001) using NAHONN



recently, the polynomial/trigonometric HONN (Lu & Zhang, 2000) and Multiple-PHONN (Zhang & Lu, 2001) models have been shown to offer improved performance, compared with P(T)HONNs.

It was mentioned previously that PHONN Model#3 comprises groups of PHONN#2 neurons. When applied to financial time-series prediction, PHONN groups produce up to an order of magnitude performance improvement over PHONNs — more specifically around 1.2% error for simulation (compared with 11%) and 5.6% error for prediction (compared with 12.7%) (Zhang, Zhang, & Fulcher, 2000). Similar improvements in performance are observed with THONN groups (Zhang, Zhang, & Fulcher, 1996, 1997).

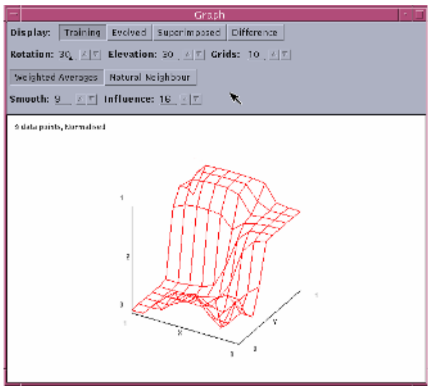
The neuron-adaptive HONN (and NAHONN group) leads to faster convergence, much reduced network size and more accurate curve fitting, compared with P(T)HONNs (Zhang, Xu, & Fulcher, 2002). Each element of the NAHONN group is a standard multilayer HONN comprising adaptive neurons, but which employs locally bounded, piecewise continuous (rather than polynomial) activation functions and thresholds.

The (1-Dimensional) neuron activation function is defined as follows:

Table 2. $A-$US exchange rate (2004)

Australian Dollar/U.S. Dollar Exchange Rate (2004)				
Month	Raw Exchange Rate	Input 1#1	Input#2	Desired Output
January	0.7576	0.95	0.93	1.00
February	0.7566	0.93	1.00	0.96
March	0.761	1.00	0.96	0.37
April	0.7584	0.96	0.37	0.03
May	0.7198	0.37	0.03	0.26
June	0.697	0.03	0.26	0.14
July	0.7125	0.26	0.14	0.00
August	0.7042	0.14	0.00	0.43
September	0.6952	0.00	0.43	0.75
October	0.7232	0.43	0.75	
November	0.7447	0.75		

Figure 13. Input data for Sigmoid PHONN simulation (prediction)



Rotation: 30 Elevation: 30 Grids: 10

Smooth: 9 Influence: 16

9 data points, Normalised

$$\psi_{i,k}(net_{i,k}) = o_{i,k}(net_{i,k}) = \sum_{h=1}^{s} f_{i,k,h}(net_{i,k}) \tag{11}$$

where $net_{i,k}$ is the input (internal state) of the *i*th neuron in the *k*th layer, and $w_{i,j,k}$ is the weight connecting the *j*th neuron in layer-(k-1) with the *i*th neuron in layer-k. This formulation, along with nD and *multi* n-Dimensional NAHONNs, incorporate free parameters which can be adjusted, along with the weights, during training (unlike conventional feed-forward ANNs). The NAHONN learning algorithm is based on steepest descent, but since the hidden-layer variables are adjustable, NAHONN offers more flexibility and more accurate approximation capability compared with (fixed activation function) MLP/BPs (Zhang, Xu, & Fulcher, 2002).

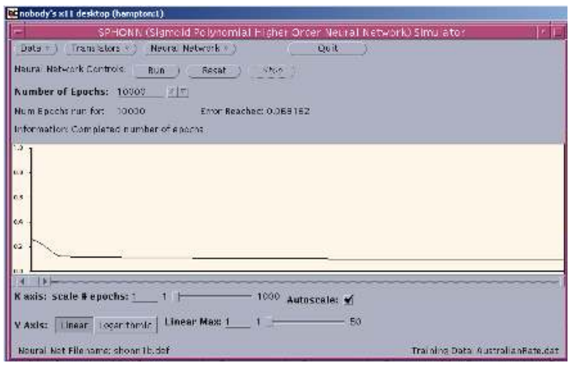
In one comparative experiment, a NAHONN with nonlinear neuron activation function led to around half the RMS error compared with PHONN, and a NAHONN which utilized piecewise NAFs required less than half the number of hidden-layer neurons, converged in less than a third of the time and led to an RMS output error two orders of magnitude lower than PHONN (Zhang, Xu, & Fulcher, 2002; Figure 12).

Now as with the earlier P(T)HONN groups, it is possible to prove a similar general result to that found previously by Hornik (1991) for ANNs, namely that NAHONN groups are capable of approximating any kind of piecewise-continuous function to any degree of accuracy (a proof is provided in Zhang, Xu, & Fulcher, 2002). Moreover, these models are capable of automatically selecting not only the optimum model for a particular time series, but also the appropriate model order.

Returning to the Sigmoid PHONN (Zhang, Crane, & Bailey, 2003), the $Australian-$US exchange rate data of Table 2 was used to predict the following month's rate — in other words, based on the previous two months rates, as follows:

$$input\#1 = \frac{current\_month - \min imum}{\max imum - \min imum} \tag{12}$$

Figure 14. Sigmoid PHONN training (convergence)


nobody's x11 desktop (hamptonc1)

SPHONN (Sigmoid Polynomial Higher Order Neural Network) Simulator

Neural Network Controls:

Number of Epochs: 10000

Num Epochs run for: 10000 Error Reached: 0.068162

Information: Completed number of epochs

X axis: Scale # epochs: 1 1 1000 Autoscale: ✓

Y Axis: Linear Logarithmic Linear Max: 1 1 50

Neural Net Filename: shoon1b.def Training Data: AustralianRate.dat

Figure 15. Sigmoid PHONN network weights

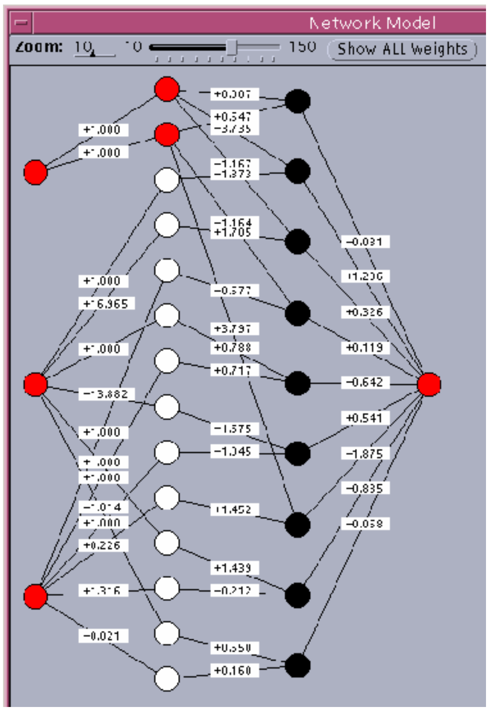
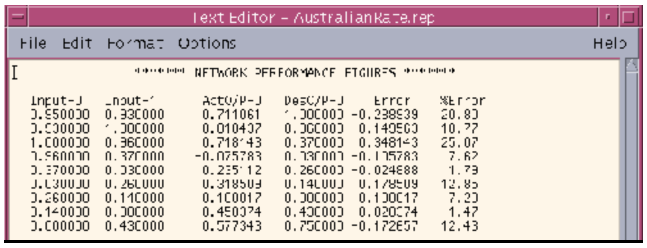


Figure 16. Sigmoid PHONN network performance


Text Editor – AustralianRate.rep
File Edit Format Options Help

```
          ******* NETWORK PERFORMANCE FIGURES *******

  Input-0   Input-1     ActO/P-0   DesO/P-0    Error     %Error
  0.550000  0.330000    0.711061   1.000000 -0.288939   28.89
  0.500000  1.000000    0.010407   0.000000  0.149500   10.77
  1.000000  0.360000    0.718143   0.370000  0.348143   25.07
  0.560000  0.370000   -0.075783   0.030000 -0.105783    7.62
  0.370000  0.030000    0.235112   0.260000 -0.024888    1.79
  0.030000  0.260000    0.318509   0.140000  0.178509   12.85
  0.260000  0.140000    0.100017   0.000000  0.100017    7.20
  0.140000  0.000000    0.450074   0.430000  0.020074    1.47
  0.000000  0.430000    0.577343   0.750000 -0.172657   12.43
```

$$input\#2 = \frac{next\_month \quad \min imum}{\max imum \quad \min imum} \tag{13}$$

$$desired\_output = \frac{month\_after\_next \quad \min imum}{\max imum \quad \min imum} \tag{14}$$

Figure 17. HONN performance comparison (Reserve Bank of Australia: Credit-card lending, August 1996-June 1997)

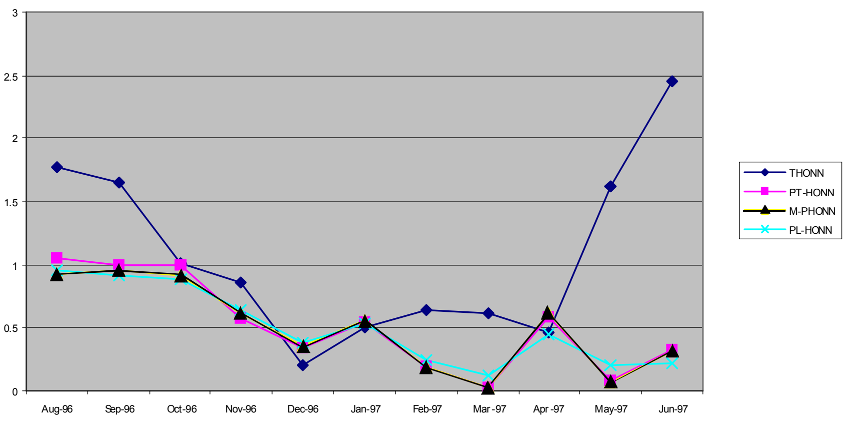
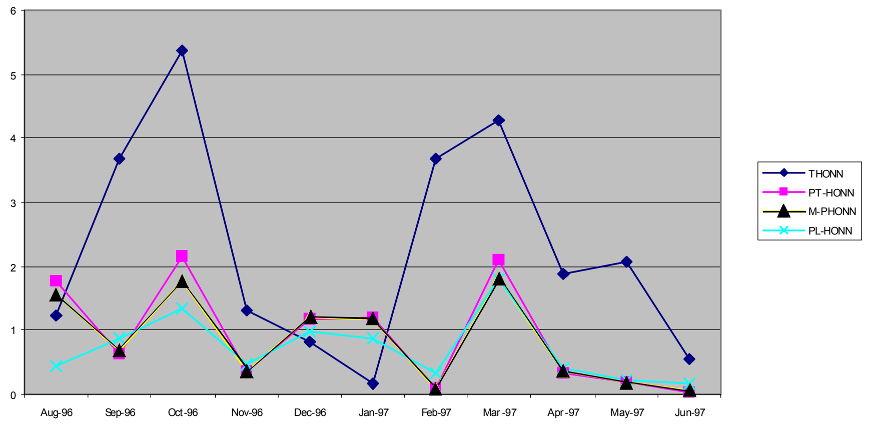


Figure 18. HONN performance comparison (Reserve Bank of Australia: Dollar-yen exchange rate (August 1996-June 1997)



Thus, only the data within the box was used for this exercise.

This input training data is plotted in Figure 13 (X = input#1, Y = input#2, and Z = desired output, respectively).

Convergence of the Sigmoid PHONN is shown in Figure 14 and the final network weights in Figure 15. In this example, convergence occurs after roughly 700 epochs, despite Figure 14 showing 10,000 epochs total (and to an error of around 7%).

In Figure 15, the third (uppermost, lefthand side) network input is the bias term, the remaining ones being input#1 (based on the current monthly rate) and input#2 (based on the next month's exchange rate); the network output is the desired (month after next) currency exchange rate, as previously explained. The grey neurons are sigmoid types, the white neurons linear, and the black ones multiplicative.

The performance of this Sigmoid PHONN (actual vs. desired outputs) is summarized in Figure 16.

More recently, Zhang (2003) has developed a *multi*PHONN, which employs a logarithmic activation function, and as its name suggests, is capable of simulating not only polynomial and/or trigonometric functions, but also combinations of these, as well as sigmoid and/or logarithmic functions. As a result, they are better able to approximate real world economic time series data. It can be seen in Figures 17 and 18 that PL-HONN offers significant performance improvement over THONNs, and marginal improvement over both PT- and M-PHONNs, when applied to typical financial time series data (Reserve Bank of Australia Bulletin: www.abs.gov.au/ausstats/abs@.nsf/w2.3).

The main finding from these experiments is that the more sophisticated PHONN variants significantly outperform THONN on typical financial time-series data, however all yield significantly lower errors compared with conventional feed-forward ANNs (not shown in this chapter's figures).

Conclusion

We have introduced the concepts of higher-order artificial neural networks and ANN groups. Such models offer significant advantages over classical feed-forward ANN models such as MLP/BP, due to their ability to better approximate complex, nonsmooth, often discontinuous training data. Important findings about the general approximation ability of such HONNs (and HONN groups) have been presented, which extend the earlier findings of Hecht-Nielsen (1987), Hornik (1991), and Leshno, Lin, Pinkus, and Schoken (1993).

Acknowledgments

We would like to acknowledge the financial assistance of the following organizations in our development of higher-order neural networks: Societe International de Telecommunications Aeronautique, Fujitsu Research Laboratories, Japan, the U.S. National Research Council, and the Australian Research Council.

References

Allen, D. W., & Taylor, J. G. (1994). Learning time series by neural networks. In M. Marinaro & P. Morasso (Eds.), *Proceedings of the International Conference on Neural Networks, Sorrento, Italy* (pp. 529-532). Berlin: Springer.

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *J. Financial Economics, 51*(2), 245-271.

Azema-Barac, M. E., & Refenes, A. N. (1997). Neural networks for financial applications. In E. Fiesler, & R. Beale (Eds.), *Handbook of neural computation* (G6.3; pp. 1-7). Oxford, UK: Oxford University Press.

Azoff, E. (1994). *Neural network time series forecasting of financial markets*. New York: Wiley.

Back, A. (2004). Independent component analysis. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 59-95). Berlin: Springer.

Barron, R., Gilstrap, L., & Shrier, S. (1987). Polynomial and neural networks: Analogies and engineering applications. In *Proceedings of the International Conference on Neural Networks, New York, Vol. 2* (pp. 431-439).

Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis: Forecasting and control* (rev. ed.). San Francisco: Holden Day.

Brockwell, P. J., & Davis, R. A. (1991). *Time series: Theory and methods* (2nd ed.). New York: Springer.

Campolucci, P., Capparelli, F., Guarnieri, S., Piazza, F., & Uncini, A. (1996). Neural networks with adaptive spline activation function. In *Proceedings of the IEEE MELECON'96 Conference*, Bari, Italy (pp. 1442-1445).

Caudill, M., & Butler, C. (1990). *Naturally intelligent systems*. Cambridge, MA: MIT Press.

Chakraborty, K., Mehrotra, K., Mohan, C., & Ranka, S. (1992). Forecasting the behaviour of multivariate time series using neural networks. *Neural Networks, 5*, 961-970.

Chang, P. T. (1997). Fuzzy seasonality forecasting. *Fuzzy Sets and Systems, 112*, 381-394.

Chatfield, C. (1996). *The analysis of time series: An introduction*. London: Chapman & Hall.

Chen, C. T., & Chang, W. D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks, 9*(4), 627-641.

Chen, L. (1981). Topological structure in visual perception. *Science, 218*, 699.

Chen, S.-H. (Ed.). (2002). *Genetic algorithms and genetic programming in computational finance*. Boston: Kluwer.

Computing Research Associates, 2005). Retrieved April 2005, from www.cra.org

Devroye, L., Gyorfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. New York: Springer.

Diamond, I., & Jeffries, J. (2001). *Beginning statistics: An introduction for social sciences.* London: Sage Publications.

Edelman, D., & Davy, P. (2004). Adaptive technical analysis in the financial markets using machine learning: A statistical view. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 1-16). Berlin: Springer.

Federal Reserve Board. (2005). Retrieved April 2005, from http://www.federalreserve.gov/releases/

Giles, L., Griffin, R., & Maxwell, T. (1988). Encoding geometric invariances in high-order neural networks. In D. Anderson (Ed.), *Proceedings Neural Information Processing Systems* (pp. 301-309).

Giles, L., & Maxwell, T. (1987). Learning, invariance and generalisation in high-order neural networks. *Applied Optics, 26*(23), 4972-4978.

Goggin, S., Johnson, K., & Gustafson, K. (1993). A second-order translation, rotation and scale invariant neural network. In R. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in neural information processing systems 3.* San Mateo, CA: Morgan Kauffman.

Gorr, W. L. (1994). Research perspective on neural network forecasting. *International Journal of Forecasting, 10*(1), 1-4.

Harris, R., & Sollis, R. (2003). *Applied time series modelling and forecasting*. Chichester, UK: Wiley.

Harvey, A. C. (1989). *Forecasting, structural times series models and the kalman filter*. Cambridge, UK: Cambridge University Press.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the International Conference on Neural Networks, Vol. 3* (pp. 11-13). New York: IEEE Press.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks, 4*, 251-257.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multi-layer feedforward networks are universal approximators. *Neural Networks, 2*, 359-366.

Hu, S., & Yan, P. (1992). Level-by-level learning for artificial neural groups. *Electronica Sinica, 20*, 10, 39-43.

Hu, Z., & Shao, H. (1992). The study of neural network adaptive control systems. *Control and Decision, 7*, 361-366.

Iba, H., & Sasaki, T. (1999). Using genetic programming to predict financial data. In *Proceedings of the 1999 Congress on Evolutionary Computation — CEC99, Vol. 1* (pp. 244-251). New Jersey: IEEE Press.

Inui, T., Tanabe, Y., & Onodera, Y. (1978). *Group theory and its application in physics*. Berlin: Springer.

Karayiannis, N., & Venetsanopoulos, A. (1993). *Artificial neural networks: Learning algorithms, performance evaluation and applications*. Boston: Kluwer.

Kingdon, J. (1997). *Intelligent systems and financial forecasting*. Berlin: Springer.

Kosmatopoulos, E. B., Polycarpou, M. M., Christodoulou, M. A., & Ioannou, P. A. (1995). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks, 6*(2), 422-431.

Lapedes, A. S., & Farber, R. (1987). Non-linear signal processing using neural networks: Prediction and system modelling. *Los Alamos National Laboratory* (Technical Report LA-UR-87).

Lee, Y. C., Doolen, G., Chen, H., Sun, G., Maxwell, T., Lee, H., et al. (1986). Machine learning using a higher order correlation network. *Physica D: Nonlinear Phenomena, 22*, 276-306.

Leshno, M., Lin, V., Pinkus, A., & Schoken, S. (1993). Multi-layer feedforward networks with a non-polynomial activation can approximate any function. *Neural Networks, 6*, 861-867.

Lippmann, R. P. (1989). Pattern classification using neural networks. *IEEE Communications Magazine, 27*, 47-64.

Lisboa, P., & Perantonis, S. (1991, November 18-21). Invariant pattern recognition using third-order networks and zernlike moments. In *Proceedings of the IEEE International Joint Conference on Neural Networks, Singapore, Vol. II* (pp. 1421-1425).

Lu, B., & Zhang, M. (2000, May 15-17). Using PT-HONN models for multi-polynomial function simulation. In *Proceedings of IASTED International Conference on Neural Networks*, Pittsburgh, PA.

Makridakis, S., Andersoen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., et al. (1982). The accuracy of extrapolation methods: Results of a forecasting competition. *Journal of Forecasting, 1*(2), 111-153.

Meuer, H., Stronmaier, E., Dongarra, J., & Simon, H. D. (2005). Retrieved April 2005, from http://www.top500.org

Mills, T. C. (1993). *The econometric modelling of financial time series*. Cambridge, UK: Cambridge University Press.

Naimark, M., & Stern, A. (1982). *Theory of group representation*. Berlin: Springer.

Plotkin, H. (1993). *Darwin machines and the nature of knowledge*. Cambridge, MA: Harvard University Press.

Psaltis, D., Park, C., & Hong, J. (1988). Higher order associative memories and their optical implementations. *Neural Networks, 1*, 149-163.

Redding, N., Kowalczyk, A., & Downs, T. (1993). Constructive higher-order network algorithm that is polynomial time. *Neural Networks, 6*, 997-1010.

Refenes, A. N. (Ed.). (1994). *Neural networks in the capital markets*. Chichester, UK: Wiley.

Reid, M. B., Spirkovska, L., & Ochoa, E. (1989). Simultaneous position, scale, rotation invariant pattern classification using third-order neural networks. *International Journal of Neural Networks, 1*, 154-159.

Reinsel, G. C. (1997). *Elements of multivariate time series analysis*. New York: Springer.

Reserve Bank of Australia Bulletin. (2005). Retrieved April 2005, from http://abs.gov.au/ausstats/

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In D. Rumelhart, & J. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1 — Foundations*. Cambridge, MA: MIT Press.

Schoneburg, E. (1990). Stock market prediction using neural networks: A project report. *Neurocomputing, 2*, 17-27.

Shin, Y., & Ghosh, J. (1991, July). The Pi-Sigma Network: An efficient higher-order neural network for pattern classification and function approximation. In *Proceedings of the International Joint Conference on Neural Networks*, Seattle, WA (pp. I: 13-18).

Sisman-Yilmaz, N. A., Alpaslan, F. N., & Jain, L. C. (2004). Fuzzy mulivariate auto-regression method and its application. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 281-300). Berlin: Springer.

Tay, E. H., & Cao, L. J. (2001). Application of support vector machines in financial time series forecasting. *Omega, 29*, 309-317.

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

Trippi, R. R., & Turban, E. (Eds.). (1993). *Neural networks in finance and investing*. Chicago: Probus.

Tseng, F. M., Tzeng, G. H., Yu, H. C., & Yuan, B. J. C. (2001). Fuzzy ARIMA model for forecasting the foreign exchange market. *Fuzzy Sets and Systems, 118*, 9-19.

University of California, Irvine. (2005). Retrieved April 2005, from http://www.ics.uci.edu/~mlearn/MLrepository.html

Vecci, L., Piazza, F., & Uncini, A. (1998). Learning and approximation capabilities of adaptive spline activation function neural networks. *Neural Networks, 11*, 259-270.

Vemuri, V., & Rogers, R. (1994). *Artificial neural networks: Forecasting time series*. Piscataway, NJ: IEEE Computer Society Press.

Watada, J. (1992). Fuzzy time series analysis and forecasting of sales volume. In J. Kacprzyk, & M. Fedrizzi (Eds.), *Studies in fuzziness Vol.1: Fuzzy regression analysis*. Berlin: Springer.

Weigend, A. S., & Gershenfeld, N. A. (Eds.). (1993). *Time series prediction: Forecasting the future and understanding the past*. Reading, MA: Addison Wesley.

Welstead, S. T. (1994). *Neural networks and fuzzy logic applications in C/C++*. New York: Wiley.

Wilcox, C. (1991). Understanding hierarchical neural network behaviour: A renormalization group approach. *Journal of Physics A, 24*, 2644-2655.

Wolpert, D. H. (1996a). The existence of a priori distinctions between learning algorithms. *Neural Computation, 8*, 1391-1420.

Wolpert, D. H. (1996b). The lack of a priori distinctions between learning algorithms. *Neural Computation, 8*, 1341-1390.

Wood, J., & Shawe-Taylor, J. (1996). A unifying framework for invariant pattern recognition. *Pattern Recognition Letters, 17*, 1415-1422.

Yamada, T., & Yabuta, T. (1992). Remarks on a neural network controller which uses an auto-tuning method for nonlinear functions. In *Proceedings of the International Joint Conference on Neural Networks, Vol. 2* (pp. 775-780).

Zeeman, E. (1962). The topology of the brain and visual perception. In M. Fork, Jr. (Ed.), *Topology of 3-manifolds and related topics*. Englewood Cliffs, NJ: Prentice Hall.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting, 14*, 35-62.

Zhang, M. (2003, May 13-15). Financial data simulation using PL-HONN Model. In *Proceedings IASTED International Conference on Modelling and Simulation*, Marina del Rey, CA (pp. 229-233).

Zhang, M., Crane, J., & Bailey, J. (2003, June 21-24). Rainfall estimation using SPHONN model. In *Proceedings of the International Conference on Artificial Intelligence*, Las Vegas, NV (pp. 695-701).

Zhang, M., & Fulcher, J. (1996). Face recognition using artificial neural network group-based adaptive tolerance (GAT) trees. *IEEE Transactions on Neural Networks, 7*(3), 555-567.

Zhang, M., & Fulcher, J. (2004). Higher order neural networks for satellite weather prediction. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 17-57). Berlin: Springer.

Zhang, M., Fulcher, J., & Scofield, R. (1997). Rainfall estimation using artificial neural network group. *Neurocomputing, 16*(2), 97-115.

Zhang, M., & Lu, B. (2001, July). Financial data simulation using M-PHONN model. In *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC (pp. 1828-1832).

Zhang, M., Murugesan, S., & Sadeghi, M. (1995, October 30-November 3). Polynomial higher order neural network for economic data simulation. In *Proceedings of the International Conference on Neural Information Processing*, Beijing, China (pp. 493-496).

Zhang, M., Zhang, J. C., & Fulcher, J. (1996, March 23-25). Financial simulation system using higher order trigonometric polynomial neural network group model. In *Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering*, New York, NY (pp. 189-194).

Zhang, M., Zhang, J. C., & Fulcher, J. (1997, June 8-12). Financial prediction system using higher order trigonometric polynomial neural network group model. In *Proceedings of the IEEE International Conference on Neural Networks*, Houston, TX (pp. 2231-2234).

Zhang, M., Zhang, J. C., & Fulcher, J. (2000). Higher-order neural network group models for data approximation. *International Journal of Neural Systems, 10*(2), 123-142.

Zhang, M., Zhang, J. C., & Keen, S. (1999, August 9-12). Using THONN system for higher frequency non-linear data simulation and prediction. In *Proceedings of the IASTED International Conference on Artificial Intelligence & Soft Computing*, Honolulu, HI (pp. 320-323).

Zhang, M., Xu, S., & Fulcher, J. (2002). Neuron-adaptive higher-order neural network models for automated financial data modelling. *IEEE Transactions on Neural Networks, 13*(1), 188-204.

Additional Sources

Machine Learning Databases: http://www.ics.uci.edu/~mlearn/MLrepository.html

Australian Bureau of Statistics, 19. Free sample data: http://www.abs.gov.au/ausstats/abs@.nsf/w2.3

National Library of Australia. Financial indicators: http://www.nla.gov.au/oz/stats.html