

## Chapter XIV

# Adaptive Higher Order Neural Network Models and Their Applications in Business

**Shuxiang Xu**

*University of Tasmania, Australia*

### ABSTRACT

*Business is a diversified field with general areas of specialisation such as accounting, taxation, stock market, and other financial analysis. Artificial Neural Networks (ANNs) have been widely used in applications such as bankruptcy prediction, predicting costs, forecasting revenue, forecasting share prices and exchange rates, processing documents and many more. This chapter introduces an Adaptive Higher Order Neural Network (HONN) model and applies the adaptive model in business applications such as simulating and forecasting share prices. This adaptive HONN model offers significant advantages over traditional Standard ANN models such as much reduced network size, faster training, as well as much improved simulation and forecasting errors, due to their ability to better approximate complex, non-smooth, often discontinuous training data sets. The generalisation ability of this HONN model is explored and discussed.*

### INTRODUCTION

Business is a diversified field with several general areas of specialisation such as accounting or financial analysis. Artificial Neural networks (ANNs) provide significant benefits in business applications. They have been actively used for applications such as bankruptcy prediction,

predicting costs, forecast revenue, processing documents and more (Kurbel et al, 1998; Atiya et al, 2001; Baesens et al, 2003). Almost any neural network model would fit into at least one business area or financial analysis. Traditional statistical methods have been used for business applications with many limitations (Azema-Barac et al, 1997; Blum et al, 1991; Park et al, 1993).

Human financial experts usually use charts of financial data and even intuition to navigate through the massive amounts of financial information available in the financial markets. Some of them study those companies that appear to be good for long-term investments. Others try to predict the future economy such as share prices based on their experiences, but with the large number of factors involved, this seems to be an overwhelming task. Consider this scenario: how can a human financial expert handle years of data for 30 factors, 500 shares, and other factors such as keeping track of the current values simultaneously? This is why some researchers insist that massive systems such as the economy of a country or the weather are not predictable due to the effects of chaos. But ANNs can be used to help automate such tasks (Zhang et al, 2002).

ANNs can be used to process subjective information as well as statistical data and are not limited to particular financial principle. They can learn from experience (existing financial data set) but they do not have to follow specific equations or rules. They can be asked to consider hundreds of different factors, which is a lot more than what human experts can digest. They won't be overwhelmed by decades of financial data, as long as the required computational power has been met. ANNs can be used together with traditional statistical methods and they do not conflict with each other (Dayhoff, 1990).

Using ANNs for financial advice means that you don't have to analyse complex financial charts in order to find a trend (of, eg, a share). The ANN architecture determines which factors correlate to each other (each factor corresponds with an input to the ANN). If patterns exist in a financial dataset, an ANN can filter out the noise and pick up the overall trends. You as the ANN program user decide what you want the ANN to learn and what kind of information it needs to be given, in order to fulfill a financial task.

ANN programs are a new computing tool which simulate the structure and operation of

the human brain. They simulate many of the human brain's most powerful abilities such as sound and image recognition, association, and more importantly the ability to generalize by observing examples (eg, forecasting based on existing situation). ANNs establish their own model of a problem based on a training process (with a training algorithm), so no programming is required because existing training programs are readily available.

Some large financial institutions have used ANNs to improve performance in such areas as bond rating, credit scoring, target marketing and evaluating loan applications. These ANN systems are typically only a few percentage points more accurate than their predecessors, but because of the amounts of money involved, these ANNs are very profitable. ANNs are now used to analyze credit card transactions to detect likely instances of fraud (Kay et al, 2006).

While conventional ANN models have been bringing huge profits to many financial institutions, they suffer from several drawbacks. First, conventional ANNs can not handle discontinuities in the input training data set (Zhang et al, 2002). Next, they do not perform well on complicated business data with high frequency components and high order nonlinearity, and finally, they are considered as 'black boxes' which can not explain their behaviour (Blum et al, 1991; Zhang et al, 2002 ; Burns, 1986).

To overcome these limitations some researchers have proposed the use of Higher Order Neural Networks (HONNs) (Redding et al, 1993 ; Zhang et al, 1999 ; Zhang et al, 2000). HONNs are able to provide some explanation for the simulation they produce and thus can be considered as 'open box' rather than 'black box'. HONNs can simulate high frequency and high order nonlinear business data, and can handle discontinuities in the input training data set (Zhang et al, 2002). Section 3 of this chapter offers more information about HONNs.

The idea of setting a few free parameters in the neuron activation function (or transfer function) of an ANN is relatively new. ANNs with such activation function seem to provide better fitting properties than classical architectures with fixed activation functions (such as sigmoid function). Such activation functions are usually called adaptive activation functions because the free parameters can be adjusted (in the same way as connection weights) to adapt to different applications. In (Vecci et al, 1998), a Feedforward Neural Network (FNN) was able to adapt its activation function by varying the control points of a Catmull-Rom cubic spline. First, this FNN can be seen as a sub-optimal realization of the additive spline based model obtained by the regularization theory. Next, simulations confirm that the special learning mechanism allows one to use the network's free parameters in a very effective way, keeping their total number at lower values than in networks with traditional fixed neuron activation functions such as the sigmoid activation function. Other notable properties are a shorter training time and a reduced hardware complexity. Based on regularization theory, the authors derived an architecture which embodies some regularity characteristics in its own activation function much better than the traditional FNN can do. Simulations on simple two-dimensional functions, on a more complex non-linear system and on a pattern recognition problem exposed the good generalization ability expected according to the theory, as well as other advantages, including the ability of tuning the activation function to determine the reduction of the number of hidden units.

Campolucci et al (1996) proposed an adaptive activation function built as a piecewise approximation with suitable cubic splines that can have arbitrary shape and allows them to reduce the overall size of the neural networks, trading connection complexity with activation function complexity. The authors developed a generalized sigmoid neural network with the adaptive activation func-

tion and a learning algorithm to operate on the identification of a non-linear dynamic system. The experimental result confirmed the computational capabilities of the proposed approach and the attainable network size reductions.

In (Chen et al, 1996), real variables  $a$  (gain) and  $b$  (slope) in the generalised sigmoid activation function were adjusted during learning process. A comparison with classical FNNs to model static and dynamical systems was reported, showing that an adaptive sigmoid (ie, a sigmoid with free parameters) leads to an improved data modelling. Based on the steepest descent method, an auto-tuning algorithm was derived to enable the proposed FNN to automatically adjust free parameters as well as connection weights between neurons. Due to the ability of auto-tuning, the flexibility and non-linearity of the FNN was increased significantly. Furthermore, the novel feature prevented the non-linear neurons from saturation, and therefore, the scaling procedure, which is usually unavoidable for traditional neuron-fixed FNNs, became unnecessary. Simulations with one and two dimensional functions approximation indicated that the proposed FNN with adaptive sigmoid activation function gave better agreement than the traditional fixed neuron FNN, even though fewer processing nodes were used. Moreover, the convergence properties were superior.

There have been limited studies with emphasis on setting free parameters in the neuron activation function before Chen and Chang (1996). To increase the flexibility and learning ability of neural networks, Kawato's group (Kawato et al, 1987) determined the near-optimal activation functions empirically. Arai et al (1991) proposed an auto-tuning method for adjusting the only free parameter in their activation function and confirmed it to be useful for image compression. Next, based on using the steepest descent method, Yamada & Yabuta (1992a,b) proposed an auto-tuning method for determining an optimal nonlinear activation function. Still, only a single

parameter that governs the shape of the nonlinear function was tuned, and their single parameter tuning method may restrict the structure of the possible optimum shape of the nonlinear activation function. Finally, Hu and Shao (1992) constructed a learning algorithm based on introducing a generalized S-shape activation function.

This chapter is organized as follows. Section 2 is a brief introduction to ANN architecture and its learning process (this section can be skipped by users who have some basic knowledge about ANN). Section 3 is a brief introduction to HONNs. Section 4 presents several Adaptive HONN models. Section 5 gives several examples to demonstrate how Adaptive HONN models can be used in business, and finally, Section 6 concludes this chapter.

## **ANN STRUCTURE AND LEARNING PROCESS (Dayhoff, 1990; Haykin, 1994; Picton, 2000)**

One of the most intriguing things about humans is how we use our brains to think, analyse, and make predictions. Our brain is composed of hundreds of billions of neurons which are massively connected with each other. Recently some biologists have discovered that it is the way the neurons are connected which gives us our intelligence, rather than what are in the neurons themselves. ANNs simulate the structure and processing abilities of the human brain's neurons and connections.

An ANN works by creating connections between different processing elements (artificial neurons), each analogous to a single neuron in a biological brain. These neurons may be physically constructed or simulated by a computer program. Each neuron takes many input signals, then, based on an internal weighting mechanism, produces a single output signal that's typically sent as input to another neuron.

The neurons are interconnected and organized into different layers. The input layer receives the

input, the output layer produces the final output. Usually one or more hidden layers are set between the two.

An ANN is taught about a specific financial problem, such as predicting a share's price, using a technique called training. Training an ANN is largely like teaching small children to remember and then recognize the letters of the English alphabet. You show a child the letter "A" and tell him what letter he's looking at. You do this a couple of times, and then ask him if he can recognise it, and if he can, you go on to the next letter. If he doesn't remember it then you tell him again that he is looking at an "A". Next, you show him a "B" and repeat the process. You would do this for all the letters of the alphabet, then start over. Eventually he will learn to recognize all of the letters of the English alphabet correctly. Later we will see that the well-known backpropagation training algorithm (supervised training) is based on this mechanism.

An ANN is fed with some financial data and it guesses what the result should be. At first the guesses would be garbage. When the trained ANN does not produce a correct guess, it is corrected. The next time it sees that data, it will guess more accurately. The network is shown lots of data (thousands of training pairs, sometimes), over and over until it learns all the data and results. Like a person, a trained ANN can generalize, which means it makes a reasonable guess when the given data have not been seen before. You decide what information to provide and the ANN finds (after learning) the patterns, trends, and hidden relationships.

The learning process involves updating the connections (usually called weights) between the neurons. The connections allow the neurons to communicate with each other and produce forecasts. When the ANN makes a wrong guess, an adjustment is made to some weights, thus it is able to learn.

ANN learning process typically begins with randomizing connection weights between the

neurons. Just like biological brains, ANNs can not do anything without learning from existing knowledge. Typically, there are two major methods for training an ANN, depending on the problem it has to solve.

A self-organizing ANN is exposed to large amounts of data and tends to discover patterns and relationships in that large data set (data mining). Researchers often use this type of training to analyze experimental data (such as economic data).

A back-propagation supervised ANN, conversely, is exposed to input-output training pairs so that a specific relationship could be learned. During the training period, the target values in the training pairs are used to evaluate whether the ANN's output is correct. If it's correct, the neural weightings that produced that output are reinforced; if the output is incorrect, those responsible weightings are diminished. This method has been extensively used by many institutions for specific problem-solving applications.

Implemented on a single computer, an ANN is usually slower than a more traditional algorithm (especially when the training set is large). The ANN's parallel nature, however, allows it to be built using multiple processors, giving it a great speed advantage at very little development cost. The parallel architecture also allows ANNs to process very large amounts of data very efficiently.

There are a few steps involved in designing a financial neural network. First of all, you need to decide what result you want the ANN to produce for you (ie, the outputs) and what information the ANN will use to arrive at the result (ie., inputs). As an example, if you want to create an ANN to predict the price of the Dow Jones Industrial Average (DOW) on a month to month average basis, one month in advance, then the inputs to the ANN would include the Consumer Price Index (CPI), the price of crude oil, the inflation rate, the prime interest rate, and others. Once these factors have been determined you then know how many input neurons should be set for the ANN. In this

example, the number of output neurons would be one because you only want to predict the price for next month. Theoretically an ANN with only one hidden layer is able to model any practical problem. The number of hidden layer neurons can not be determined based on any universal rules but generally speaking this number should be less than  $N/d$ , where  $N$  is the number of training data sets and  $d$  is the number of input neurons (Barron, 1994).

It's usually good to give the ANN lots of information. If you are unsure if a factor is related to the output, the neural network will determine if the factor is important and will learn to ignore anything irrelevant. Sometimes a possibly irrelevant piece of information can allow the ANN to make distinctions which we are not aware of (which is the essence of data mining). If there's no correlation, the ANN will just ignore the factor.

## **HONNs**

HONNs (Higher Order Neural Networks) (Lee et al, 1986) are networks in which the net input to a computational neuron is a weighted sum of products of its inputs. Such neuron is called a Higher-order Processing Unit (HPU) (Lippman, 1989). It was known that HONN's can implement invariant pattern recognition (Psaltis et al, 1988 ; Reid et al, 1989 ; Wood et al, 1996). Giles in (Giles et al, 1987) showed that HONN's have impressive computational, storage and learning capabilities. In (Redding et al, 1993), HONN's were proved to be at least as powerful as any other FNN architecture when the orders of the networks are the same. Kosmatopoulos et al (1995) studied the approximation and learning properties of one class of recurrent HONNs and applied these architectures to the identification of dynamical systems. Thimm et al (1997) proposed a suitable initialization method for HONN's and compared this method to weight initialization techniques



for FNNs. A large number of experiments were performed which led to the proposal of a suitable initialization approach for HONNs.

Giles et al (1987) showed that HONN's have impressive computational, storage and learning capabilities. The authors believed that the order or structure of a HONN could be tailored to the order or structure of a particular problem, and thus a HONN designed for a particular class of problems becomes specialized and efficient in solving these problems. Furthermore, a priori knowledge could be encoded in a HONN.

In Redding et al (1993), HONN's were proved to be at least as powerful as any other FNN architecture when the order of the networks are the same. A detailed theoretical development of a constructive, polynomial-time algorithm that would determine an exact HONN realization with minimal order for an arbitrary binary or bipolar mapping problem was created to deal with the two-or-more clumps problem, demonstrating that the algorithm performed well when compared with the Tiling and Upstart algorithms.

Kosmatopoulos et al (1995) studied the approximation and learning properties of one class of recurrent HONN's and applied these architectures to the identification of dynamical systems. In recurrent HONN's the dynamic components are distributed throughout the network in the form of dynamic neurons. It was shown that if enough higher order connections were allowed, then this network was capable of approximating arbitrary dynamical systems. Identification schemes based on higher order network architectures were designed and analyzed.

Thimm et al (1997) proposed a suitable initialization method for HONN's and compared this method to weight initialization techniques for FNN's. As proper initialization is one of the most important prerequisites for fast convergence of FNN's, the authors aimed at determining the optimal variance (or range) for the initial weights and biases, the principal parameters of random initialization methods. A large number

of experiments were performed which led to the proposal of a suitable initialization approach for HONN's. The conclusions were justified by sufficiently small confidence intervals of the mean convergence times.

## **ADAPTIVE HONN MODELS**

Adaptive HONNs are HONNs with adaptive activation functions. The network structure of an Adaptive HONN is the same as that of a multi-layer FNN. That is, it consists of an input layer with some input units, an output layer with some output units, and at least one hidden layer consisting of intermediate processing units. Usually there is no activation function for neurons in the input layer and the output neurons are summing units (linear activation), and the activation function in the hidden units is an adaptive one.

In (Zhang et al, 2002) a one-dimensional Adaptive HONN was defined as follows.

Suppose that:

$i$  = The  $i$ th neuron in layer- $k$

$k$  = The  $k$ th layer of the neural network

$h$  = The  $h$ th term in the *NAF* (Neural network Activation Function)

$s$  = The maximum number of terms in the *NAF*

$x$  = First neural network input

$y$  = Second neural network input

$net_{i,k}$  = The input or internal state of the  $i$ th neuron in the  $k$ th layer

$w_{i,j,k}$  = The weight that connects the  $j$ th neuron in layer  $k - 1$  with the  $i$ th neuron in layer  $k$

$o_{i,k}$  = The value of the output from the  $i$ th neuron in layer -  $k$

The one-dimension adaptive HONN activation function is defined as:

$$NAF: \Psi_{i,k}(net_{i,k}) = o_{i,k}(net_{i,k}) = \sum_{h=1}^s f_{i,k,h}(net_{i,k}) \quad (1)$$

In case of  $s = 4$ :

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

The one-dimensional Adaptive HONN activation function then becomes Equation (3), where  $a1_{i,k}, b1_{i,k}, c1_{i,k}, a2_{i,k}, b2_{i,k}, a3_{i,k}, a4_{i,k}, b4_{i,k}$  are free parameters which can be adjusted (as well as weights) during training.

In this chapter, we will only discuss the following special case of the above adaptive activation function:  $a1_{i,k} = b1_{i,k} = 0, a4_{i,k} = b4_{i,k} = 0$ .

So the adaptive activation function we are interested in is Equation (4).

The learning algorithm for Adaptive HONN with activation function (4) can be found in the Appendix section.

## ADAPTIVE HONN MODEL APPLICATIONS IN BUSINESS

In this section, the Adaptive HONN model as defined in Section 4 has been used in several

financial applications. The results are given and discussed.

### Simulating and Forecasting Total Taxation Revenues of Australia

The Adaptive HONN model has been used to simulate and forecast the Total Taxation Revenues of Australia as shown in Figure 5.1. The financial data were downloaded from the Australian Taxation Office (ATO) web site. For this experiment monthly data between Jan 1994 and Dec 1999 were used. The detailed comparison between the adaptive HONN and traditional standard ANN for this example is illustrated in Table 5.1.

After the Adaptive HONN (with only 4 hidden units) has been well trained over the training data pairs, it was used to forecast the taxation revenues for each month of the year 2000. Then the forecasted revenues were compared with the real revenues for the period and the overall RMS error reached 2.55%. To demonstrate the advantages of the Adaptive HONN, the above-trained Standard ANN (with 18 hidden units) was also used for the same forecasting task which resulted in an overall RMS error of 5.63%.

Next, some cross-validation approach was used to improve the performance of the Adaptive HONN. Cross-validation is the statistical practice of dividing a sample of data into subsets so that the experiment is initially performed on a single

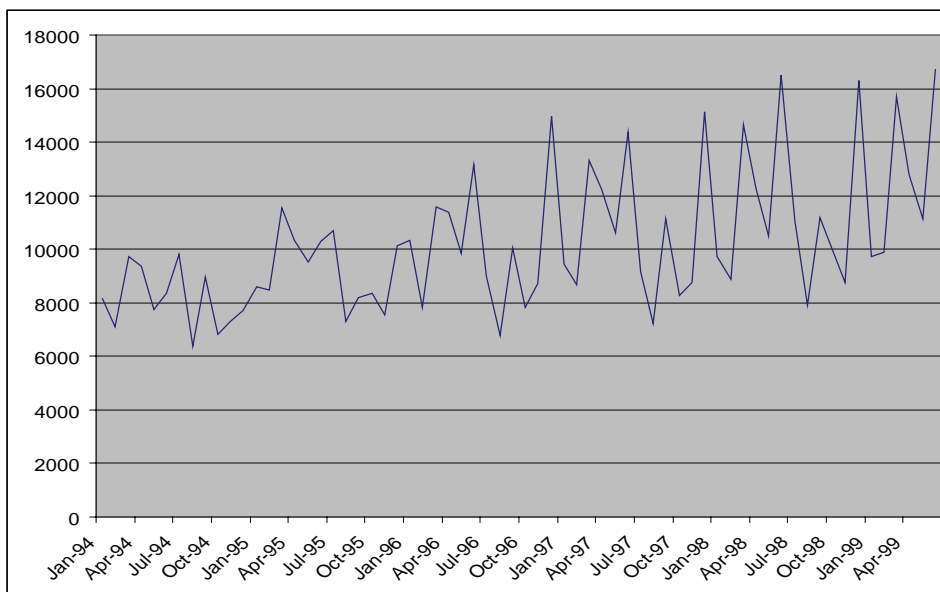
Equation (3).

$$\begin{aligned} \Psi_{i,k}(net_{i,k}) &= \sum_{h=1}^4 f_{i,k,h}(net_{i,k}) \\ &= a1_{i,k} \cdot \sin^{c1_{i,k}}(b1_{i,k} \cdot (net_{i,k})) + a2_{i,k} \cdot e^{-b2_{i,k} \cdot (net_{i,k})^2} + a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (net_{i,k})}} + a4_{i,k} \cdot (net_{i,k})^{b4_{i,k}} \end{aligned}$$

Equation (4).

$$\Psi_{i,k}(net_{i,k}) = \sum_{h=1}^2 f_{i,k,h}(net_{i,k}) = a2_{i,k} \cdot e^{-b2_{i,k} \cdot (net_{i,k})^2} + a3_{i,k} \cdot \frac{1}{1 + e^{-b3_{i,k} \cdot (net_{i,k})}}$$

*Figure 5.1. Total taxation revenues of Australia (\$ million) (Jan 1994 To Dec 1999)*



*Table 5.1. Adaptive HONN with NAF and standard ANN to simulate taxation revenues*

Neural Network	No. HL	HL Nodes	Epoch	RMS Error
Adaptive HONN	1	4	5,000	0.025468
Standard ANN	1	4	12,000	0.965874
Standard ANN	1	10	12,000	0.856654
Standard ANN	1	14	12,000	0.087996
Standard ANN	1	18	12,000	0.056345

(HL: Hidden Layer. RMS: Root-Mean-Square)

subset, while the other subset(s) are retained for subsequent use in confirming and validating the initial analysis. The initial subset of data is usually called the training set, and the other subset(s) are called validation or testing sets. Cross-validation is one of several approaches for estimating how well the ANN you've just trained from some training data is going to perform on future as-yet-unseen data. Cross-validation can be used to estimate the generalization error of a given ANN model. It can also be used for model selection by choosing one of several models that has the smallest estimated generalization error.

For this example, the training data set was divided into a training set made 70% of the original training set and a validation set made of 30% of the original training set. The training (training time and number of epochs) was optimized based on evaluation over the validation set. Then the well-trained Adaptive HONN was used to forecast the taxation revenues for each month of the year 2000, and the forecasted taxation revenues were compared with the real prices for the period. The overall RMS error reached 2.05%. The same mechanism was applied to using a Standard ANN, which resulted in an RMS error of 4.77%.



Figure 5.2. Reserve Bank Of Australia Assets (\$ million) (Jan 1980 To Dec 2000)

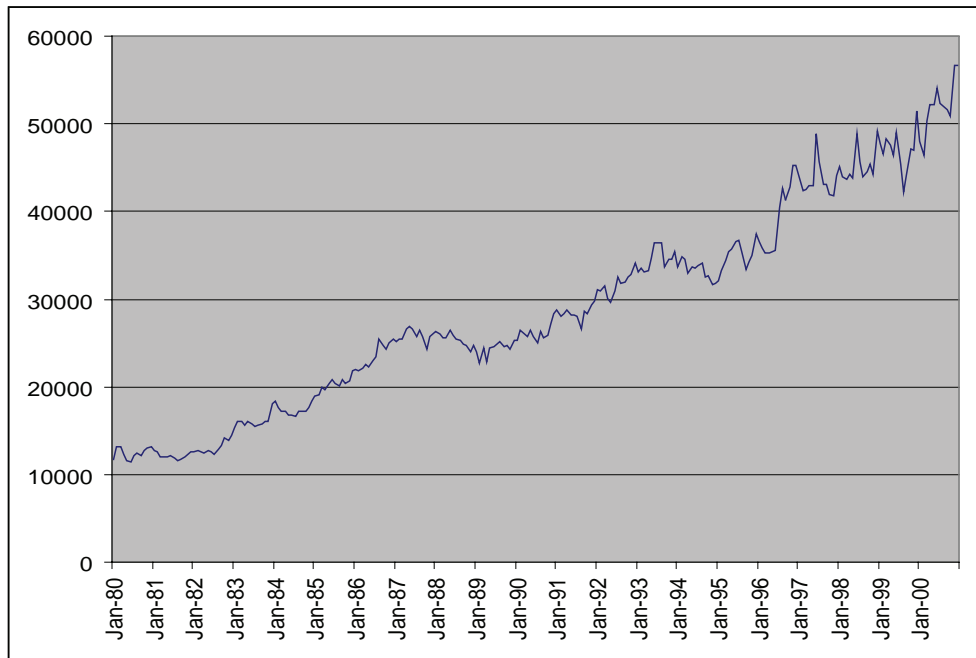


Table 5.2. Adaptive HONN with NAF and standard ANN to simulate Reserve Bank Of Australia Assets (\$ million)

Neural Network	No. HL	HL Nodes	Epoch	RMS Error
Adaptive HONN	1	3	5,000	0.019654
Standard ANN	1	3	12,000	0.912354
Standard ANN	1	9	12,000	0.798652
Standard ANN	1	16	12,000	0.065487
Standard ANN	1	22	12,000	0.053321

(HL: Hidden Layer. RMS: Root-Mean-Square)

### Simulating and Forecasting Reserve Bank Of Australia Assets

The Adaptive HONN model has also been used to simulate and forecast the Reserve Bank Of Australia Assets as shown in Figure 5.2. The financial data were obtained from the Reserve Bank Of Australia. For this experiment monthly data between Jan 1980 and Dec 2000 were used. The detailed comparison between the adaptive HONN and traditional standard ANN for this example is illustrated in Table 5.2.

After the Adaptive HONN (with only 3 hidden units) has been well trained over the training data pairs, it was used to forecast the Reserve Bank Of Australia Assets for each month of the year 2001. Then the forecasted assets were compared with the real assets for the period and the overall RMS error reached 1.96%. To demonstrate the advantages of the Adaptive HONN, the above-trained Standard ANN (with 22 hidden units) was also used for the same forecasting task which resulted in an overall RMS error of 5.33%.

Again, some cross-validation approach was used to improve the performance of the Adaptive

HONN. For this time, the training data set was divided into a training set made 75% of the original training set and a validation set made of 25% of the original training set. The training (training time and number of epochs) was optimized based on evaluation over the validation set. Then the well-trained Adaptive HONN was used to predict the assets for each month of the year 2001, and the forecasted assets were compared with the real assets for the period. The overall RMS error reached 1.80%. The same mechanism was applied to using a Standard ANN, which resulted in an RMS error of 5.02%.

### **Simulating and Forecasting Fuel Economy**

In the next experiment a dataset containing information of different cars built in the US, Europe, and Japan was trained using the Adaptive HONN to determine car fuel economy (MPG - Miles Per Gallon) for each vehicle. There were a total of 392 samples in this data set with 9 input variables and 1 output. The dataset was from UCI Machine Learning Repository (2007). The output was the fuel economy in MPG, and the input variables were:

- Number of cylinders
- Displacement
- Horsepower
- Weight
- Acceleration
- Model year
- Made in US? (0,1)
- Made in Europe? (0,1)
- Made in Japan? (0,1)

To compare the performance of Adaptive HONN and Standard ANN the dataset was divided into a set containing 353 samples for training, and a set containing 39 samples for forecasting (or generalization). This time, a cross-validation mechanism was adopted directly which split the

training set into 2 sections to train both an Adaptive HONN and a Standard ANN. After both neural networks were well-trained, the forecasting RMS error (over the 39 samples) from the Adaptive HONN reached 6.03%, while the forecasting error from the Standard ANN (over the 39 samples) reached 13.55%.

### **CONCLUSION**

In this chapter an Adaptive HONN model was introduced and applied in business applications such as simulating and forecasting government taxation revenues. Such models offer significant advantages over traditional Standard ANN models such as much reduced network size, faster training, as well as much improved simulation and forecasting errors, due to their ability to better approximate complex, non-smooth, often discontinuous training data sets. Compared with some existing approaches on applying ANN models in business applications, although there are more free parameters in the Adaptive HONN model, training speed is increased due to a significant decrease of network size. What is more, simulation and forecasting accuracy is greatly improved, which has to be one of the main concerns in the business world.

The method described in this chapter relies on using cross-validation to improve generalisation ability of the Adaptive HONN model. As part of the future research, some current cross-validation approaches would be improved so that the forecasting errors could be reduced further down to a more satisfactory level. More factors which can help improve the generalisation ability would be considered.

### **ACKNOWLEDGMENT**

The author wishes to thank Prof Ming Zhang for his valuable advice on this chapter.

## REFERENCES

- Arai, M., Kohon, R., & Imai, H. (1991). Adaptive control of a neural network with a variable function of a unit and its application, *Transactions on Inst. Electronic Information Communication Engineering*, J74-A, 551-559.
- Atiya, A.F. (2001). Bankruptcy prediction for credit risk using neural networks: A survey and new results. *IEEE Transactions on Neural Networks*, 12(4), 929-935.
- Azema-Barac, M., & Refenes, A. (1997). Neural networks for financial applications. In Fiesler, E., & Beale, R. (Eds.), *Handbook of Neural Computation*. Oxford University Press (G6:3:1-7).
- Baesens, B., Setiono, R., Mues, C., & Vanthienen, J. (2003). Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science*, 49(3).
- Barron, A. R. (1994). Approximation and estimation bounds for artificial neural networks. *Machine Learning*, (14), 115-133.
- Blum, E., & Li, K. (1991). Approximation theory and feed-forward networks. *Neural Networks*, 4, 511-515.
- Burns, T. (1986). The interpretation and use of economic predictions. *Proc. Royal Society A*, pp. 103-125.
- Campolucci, P., Capparelli, F., Guarnieri, S., Piazza, F., & Uncini, A. (1996). Neural networks with adaptive spline activation function. *Proceedings of IEEE MELECON 96* (pp. 1442-1445). Bari, Italy.
- Chen, C.T., & Chang, W.D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks*, 9(4), 627-641.
- Dayhoff, J. E. (1990). *Neural network architectures : An introduction*. New York: Van Nostrand Reinhold.
- Gallant, A. R., & White, H. (1988). There exists a neural network that does not make avoidable mistakes. *IEEE Second International Conference on Neural Networks, I*, 657-664. San Diego: SOS Printing,
- Giles, C.L., & Maxwell, T. (1987). Learning, invariance, and generalization in higher order neural networks. *Applied Optics*, 26(23), 4972-4978.
- Grossberg, S. (1986). Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity. *Proc. National Academy of Sciences*, 59, 368-372.
- Hammadi, N. C., & Ito, H. (1998). On the activation function and fault tolerance in feedforward neural networks. *IEICE Transactions on Information & Systems*, E81D(1), 66 – 72.
- Hansen, J.V., & Nelson, R.D. (1997). Neural networks and traditional time series methods: A synergistic combination in state economic forecasts. *IEEE Transactions on Neural Networks*, 8(4), 863-873.
- Hinton, G. E. (1989). Connectionist learning procedure, *Artificial Intelligence*, 40, 251 – 257.
- Holden, S.B., & Rayer, P.J.W. (1995). Generalisation and PAC learning: some new results for the class of generalised single-layer networks. *IEEE Transactions on Neural Networks*, 6(2), 368 – 380.
- Haykin, S. S. (1994). *Neural networks : A comprehensive foundation*. New York : Macmillan.
- Hu, Z., & Shao, H. (1992). The study of neural network adaptive control systems. *Control and Decision*, 7, 361-366.
- Kawato, M., Uno, Y., Isobe, M., & Suzuki, R. (1987) A hierarchical model for voluntary movement and its application to robotics. *Proc. IEEE Int. Conf. Network*, IV, 573-582.
- Kay, A. (2006). Artificial neural networks. *Computerworld*. Retrieved on 27 November 2006 from

<http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,57545,00.html>

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.

Kurbel, K., Singh, K., & Teuteberg, F. (1998). Search and classification of interesting business applications in the World Wide Web using a neural network approach. *Proceedings of the 1998 IACIS Conference*. Cancun, Mexico.

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

Lippman, R.P. (1989). Pattern classification using neural networks. *IEEE Commun. Mag.*, 27, 47-64.

Park, J., & Sandberg, I.W. (1993). Approximation and radial-basis-function networks. *Neural Computation*, 5, 305-316.

Picton, P. (2000). *Neural networks*. Basingstoke: Palgrave.

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

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

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

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

Rumelhart, D.E., & McClelland, J.L. (1986). *Parallel distributed computing: Exploration in the microstructure of cognition*. Cambridge, MA: MIT Press.

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

*UCI Machine Learning Repository* (2007). Retrieved April 2007 from <ftp://ftp.ics.uci.edu/pub/machine-learning-databases/auto-mpg/auto-mpg.data>

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

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. *IJCNN*, 2, 775-780.

Zhang, M., Xu, S., & Lu B. (1999). Neuron-adaptive higher order neural network group models. *Proc. Intl. Joint Conf. Neural Networks - IJCNN'99*, Washington, DC, USA, (Paper # 71).

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

Zhang, M., Zhang, J. & Fulcher, J. (2000). Higher order neural network group models for financial simulation. *Intl. J. Neural Systems*, 12(2), 123-142.

## **ADDITIONAL READING**

Baptista-Filho, B. D., Cabral, E. L. L., & Soares, A. J. (1999). A new approach to artificial neural

- networks. *IEEE Transactions on Neural Networks*, 9(6), 1167 – 1179.
- Barron, A. (1993). Universal approximation bounds for superposition of a sigmoidal function. *IEEE Transactions on Information Theory*, 3, 930-945.
- Brent, R. P. (1991). Fast training algorithm for multilayer neural networks. *IEEE Transactions on Neural Networks*, 2, 346 – 354.
- Carroll, S., & Dickinson, B. (1989). Construction of neural networks using the radon transform. *IEEE International Conference on Neural Networks*, Vol. 1, pp. 607 – 611. Washington DC.
- Cichocki, A., & Unbehauen, R. (1993). *Neural networks for optimization and signal processing*. New York: Wiley.
- Clemen, R.T. (1989). Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5, 559 – 583.
- Day, S., & Davenport, M. (1993). Continuous-time temporal backpropagation with adaptive time delays. *IEEE Transactions on Neural Networks*, 4, 348 – 354.
- Durbin, R., & Rumelhart, D. E. (1989). Product units: a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1, 133 – 142.
- Finnoff, W., Hergent, F., & Zimmermann, H.G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, 6, 771 - 783.
- Fogel, D.B. (1991). *System identification through simulated evolution: A machine learning approach to modelling*. Needham Heights, MA: Ginn.
- Gallant, S.I. (1993). *Neural Network Learning and Expert Systems*. Cambridge, MA: MIT Press.
- Geva, S., & Sitte, J. (1992). A constructive method for multivariate function approximation by multilayered perceptrons. *IEEE Transactions on Neural Networks*, 3(4), 621-623.
- Girosi, F., Jones, M., & Poggio, T. (1995). Regularisation theory and neural networks architecture. *Neural Computation*, 7, 219 – 269.
- Gorr, W.L. (1994). Research prospective on neural network forecasting. *International Journal of Forecasting*, 10(1), 1-4.
- Grossberg, S. (1976). Adaptive pattern classification and universal recording. I: Parallel development and coding of neural detectors. *Biological Cybernetics*, 23, 121-134.
- Harp, S., Samad, T., & Guuha, A. (1989). Toward the genetic synthesis of neural networks. In D. Shaffer (Ed.), *Proceedings of 3rd International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.
- Hill, T., Marquez, L., O'Connor, M., & Remus, W. (1994). Artificial neural network models for forecasting and decision making. *International Journal of Forecasting*, 10, 5 – 15.
- Hill, T., O'Connor, M., & Remus, W. (1996). Neural network models for time series forecasting. *Management Science*, 42, 1082 – 1092.



## APPENDIX

We use the following notations:

$I_{i,k}(u)$	the input or internal state of the $i$ th neuron in the $k$ th layer
$w_{i,j,k}$	the weight that connects the $j$ th neuron in layer $k-1$ and the $i$ th neuron in layer $k$
$O_{i,k}(u)$	the value of output from the $i$ th neuron in layer $k$
$A1, B1, A2, B2$	adjustable variables in activation function
$\theta_{i,k}$	the threshold value of the $i$ th neuron in the $k$ th layer
$d_j(u)$	the $j$ th desired output value
$\beta$	learning rate
$m$	total number of output layer neurons
$l$	total number of network layers
$r$	the iteration number
$\eta$	momentum

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

$$I_{i,k}(u) = \sum_j [w_{i,j,k} O_{j,k-1}(u)] - \theta_{i,k} \quad (\text{A.1})$$

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

$$O_{i,k}(u) = \Psi(I_{i,k}(u)) = A1_{i,k} \cdot e^{-B1_{i,k} \cdot I_{i,k}(u)} + \frac{A2_{i,k}}{1 + e^{-B2_{i,k} \cdot I_{i,k}(u)}} \quad (\text{A.2})$$

To train our neural network an energy function:

$$E = \frac{1}{2} \sum_{j=1}^m (d_j(u) - O_{j,l}(u))^2 \quad (\text{A.3})$$

is adopted, which is the sum of the squared errors between the actual network output and the desired output for all input patterns. In (A.3),  $m$  is the total number of output layer neurons,  $l$  is the total number of constructed network layers (here  $l = 3$ ). The aim of learning is undoubtedly to minimize the energy function by adjusting the weights associated with various interconnections, and the variables in the activation function. This can be fulfilled by using a variation of the steepest descent gradient rule (Rumelhart et al, 1986) expressed as follows:

$$w_{i,j,k}^{(r)} = \eta w_{i,j,k}^{(r-1)} + \beta \frac{\partial E}{\partial w_{i,j,k}} \quad (\text{A.4})$$

$$\theta_{i,k}^{(r)} = \eta \theta_{i,k}^{(r-1)} + \beta \frac{\partial E}{\partial \theta_{i,k}} \quad (\text{A.5})$$

$$A1_{i,k}^{(r)} = \eta A1_{i,k}^{(r-1)} + \beta \frac{\partial E}{\partial A1_{i,k}} \quad (\text{A.6})$$

$$B1_{i,k}^{(r)} = \eta B1_{i,k}^{(r-1)} + \beta \frac{\partial E}{\partial B1_{i,k}} \quad (\text{A.7})$$

$$A2_{i,k}^{(r)} = \eta A2_{i,k}^{(r-1)} + \beta \frac{\partial E}{\partial A2_{i,k}} \quad (\text{A.8})$$

$$B2_{i,k}^{(r)} = \eta B2_{i,k}^{(r-1)} + \beta \frac{\partial E}{\partial B2_{i,k}} \quad (\text{A.9})$$

To derive the gradient information of  $E$  with respect to each adjustable parameter in equations (A.4)-(A.9), we define:

$$\frac{\partial E}{\partial I_{i,k}(u)} = \zeta_{i,k} \quad (\text{A.10})$$

$$\frac{\partial E}{\partial O_{i,k}(u)} = \xi_{i,k} \quad (\text{A.11})$$

Now, from equations (A.2), (A.3), (A.10) and (A.11), we have the partial derivatives of  $E$  with respect to adjustable parameters as follows:

$$\frac{\partial E}{\partial w_{i,j,k}} = \frac{\partial E}{\partial I_{i,k}(u)} \frac{\partial I_{i,k}(u)}{\partial w_{i,j,k}} = \zeta_{i,k} O_{j,k-1}(u) \quad (\text{A.12})$$

$$\frac{\partial E}{\partial \theta_{i,k}} = \frac{\partial E}{\partial I_{i,k}(u)} \frac{\partial I_{i,k}(u)}{\partial \theta_{i,k}} = -\zeta_{i,k} \quad (\text{A.13})$$

$$\frac{\partial E}{\partial A1_{i,k}} = \frac{\partial E}{\partial O_{i,k}} \frac{\partial O_{i,k}}{\partial A1_{i,k}} = \xi_{i,k} e^{-B1_{i,k} \cdot I_{i,k}} \quad (\text{A.14})$$

$$\begin{aligned} \frac{\partial E}{\partial B1_{i,k}} &= \frac{\partial E}{\partial O_{i,k}} \frac{\partial O_{i,k}}{\partial B1_{i,k}} \\ &= -\xi_{i,k} \cdot A1_{i,k} \cdot I_{i,k} \cdot e^{-B1_{i,k} \cdot I_{i,k}} \end{aligned} \quad (\text{A.15})$$

$$\frac{\partial E}{\partial A2_{i,k}} = \frac{\partial E}{\partial O_{i,k}} \frac{\partial O_{i,k}}{\partial A2_{i,k}} = \xi_{i,k} \cdot \frac{1}{1 + e^{-B2_{i,k} \cdot I_{i,k}}} \quad (\text{A.16})$$

$$\frac{\partial E}{\partial B2_{i,k}} = \frac{\partial E}{\partial O_{i,k}(u)} \frac{\partial O_{i,k}(u)}{\partial B2_{i,k}} = \xi_{i,k} \cdot \frac{A2_{i,k} \cdot I_{i,k}(u) \cdot e^{-B2_{i,k} \cdot I_{i,k}(u)}}{(1 + e^{-B2_{i,k} \cdot I_{i,k}(u)})^2} \quad (\text{A.17})$$

And for (A.10) and (A.11) the following equations can be computed:

$$\zeta_{i,k} = \frac{\partial E}{\partial I_{i,k}} = \frac{\partial E}{\partial O_{i,k}} \frac{\partial O_{i,k}}{\partial I_{i,k}} = \xi_{i,k} \cdot \frac{\partial O_{i,k}}{\partial I_{i,k}} \quad (\text{A.18})$$

while:

$$\frac{\partial O_{i,k}(u)}{\partial I_{i,k}(u)} = A1_{i,k} \cdot B1_{i,k} \cdot e^{-B1_{i,k} \cdot I_{i,k}(u)} + \frac{A2_{i,k} \cdot B2_{i,k} \cdot e^{-B2_{i,k} \cdot I_{i,k}(u)}}{(1 + e^{-B2_{i,k} \cdot I_{i,k}(u)})^2} \quad (\text{A.19})$$

and:

$$\xi_{i,k} = \begin{cases} \sum_j \zeta_{j,k+1} w_{j,i,k+1}, & \text{if } 1 \leq k < l; \\ O_{i,l} - d_i, & \text{if } k = l. \end{cases} \quad (\text{A.20})$$

All 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.