ELSEVIER

Neurocomputing 10 (1996) 215–236

NEUROCOMPUTING

Designing a neural network for forecasting financial and economic time series

Iebeling Kaastra [a], Milton Boyd [b,*]

[a] *Manager, Pricing, Canadian Wheat Board, 423 Main Street, P.O. Box 816, Winnipeg R3C 2P5, Canada*
[b] *Associate Professor, University of Manitoba, 403-66 Dafoe Rd., Winnipeg R3T 2N2, Canada*

Received 31 August 1994; accepted 23 March 1995

Abstract

Artificial neural networks are universal and highly flexible function approximators first used in the fields of cognitive science and engineering. In recent years, neural network applications in finance for such tasks as pattern recognition, classification, and time series forecasting have dramatically increased. However, the large number of parameters that must be selected to develop a neural network forecasting model have meant that the design process still involves much trial and error. The objective of this paper is to provide a practical introductory guide in the design of a neural network for forecasting economic time series data. An eight-step procedure to design a neural network forecasting model is explained including a discussion of tradeoffs in parameter selection, some common pitfalls, and points of disagreement among practitioners.

Keywords: Neural networks; Financial forecasting; Backpropagation; Data preprocessing; Training; Testing; Validation; Network paradigms; Learning rate; Momentum and forecast evaluation criteria

1. Introduction

Neural networks have become increasingly popular in finance as financial services organizations have been the second largest sponsors of research in neural network applications [28]. Typical applications in finance include risk rating of mortgages and fixed income investments, index construction, simulation of market

* Communicating author. Fax: 204-983-4993, email: mboyd@bldgagric.lan1.umanitoba.ca.

SSDI 0925-2312(95)00039-9

Table 1
Common parameters in designing a backpropagation neural network

Data preprocessing
frequency of data - daily, weekly, monthly, quarterly
type of data - technical, fundamental
method of data sampling
method of data scaling - maximum/minimum, mean/standard deviation
Training
learning rate per layer
momentum term
training tolerance
epoch size
learning rate limit
maximum number of runs
number of times to randomize weights
size of training, testing, and validation sets
Topology
number of input neurons
number of hidden layers
number of hidden neurons in each layer
number of output neurons
transfer function for each neuron
error function

behaviour, portfolio selection/diversification, identification of economic explanatory variables, and economic forecasting [27].

Neural networks are universal function approximators that can map any nonlinear function [29]. As such flexible function approximators, they are powerful methods for pattern recognition, classification, and forecasting. Neural networks are less sensitive to error term assumptions and they can tolerate noise, chaotic components, and heavy tails better than most other methods [18]. Other advantages include greater fault tolerance, robustness, and adaptability compared to expert systems due to the large number of interconnected processing elements that can be 'trained' to learn new patterns [28,17].

In practice, it appears that although many organizations have expressed interest in applying neural network technology, few have actually implemented them successfully [7]. Those that have been successful have spent considerable resources experimenting and fine tuning neural networks for their particular applications as evidenced by the many commercial neural network software packages originally developed as in-house proprietary programs.

Neural networks have been criticized and their wide-spread successful application likely hindered because of the black box nature of their solutions, excessive training times, difficulty in obtaining and later replicating a stable solution, the danger of overfitting, tedious software, and the large number of parameters that must be experimentally selected to generate a good forecast. Table 1 lists the most common parameters that a researcher must choose when designing a neural

network forecasting model. It excludes the many different proprietary features offered by neural network software vendors and ignores some more advanced topics. The large number of ways to organize a neural network account for its powerful function approximation capabilities. The cost of such flexibility in modeling time series data is that the researcher must select the right combination of parameters. As a result of the large number of parameters and the relatively recent introduction of neural networks to finance, deciding on the appropriate network paradigm still involves much trial and error.

Therefore, the objective of this paper is to provide an overview of a step by step methodology to design a neural network for forecasting economic time series data. First, the architecture of a backpropagation (BP) neural network is briefly discussed. The BP network is used to illustrate the design steps since it is capable of solving a wide variety of problems and it is the most common type of neural network in time series forecasting. This is followed by an explanation of an eight-step procedure for designing a neural network including a discussion of tradeoffs in parameter selection, some common pitfalls, and points of disagreement among practitioners. While there are few hard rules, the literature does contain numerous rules of thumb and practical advice that can assist beginners in designing a successful neural network forecasting model.

2. Backpropagation neural networks

Backpropagation (BP) neural networks consist of a collection of inputs and processing units known as either neurons, neurodes, or nodes (Fig. 1). The neurons in each layer are fully interconnected by connection strengths called weights which, along with the network architecture, store the knowledge of a trained network. In addition to the processing neurons, there is a bias neuron connected to each processing unit in the hidden and output layers. The bias neuron has a value of positive one and serves a similar purpose as the intercept in regression models. The neurons and bias terms are arranged into layers; an input layer, one or more hidden layers, and an output layer. The number of hidden layers and neurons within each layer can vary depending on the size and nature of the data set.

Neural networks are similar to linear and non-linear least squares regression and can be viewed as an alternative statistical approach to solving the least squares problem [30]. Both neural networks and conventional regression analysis attempt to minimize the sum of squared errors. The bias term is analogous to the intercept term in a regression equation. The number of input neurons is equal to the number of independent variables while the output neuron(s) represent the dependent variable(s). Linear regression models may be viewed as a feedforward neural network with no hidden layers and one output neuron with a linear transfer function. The weights connecting the input neurons to the single output neuron are analogous to the coefficients in a linear least squares regression. Networks with one hidden layer resemble nonlinear regression models. The weights represent regression curve parameters.

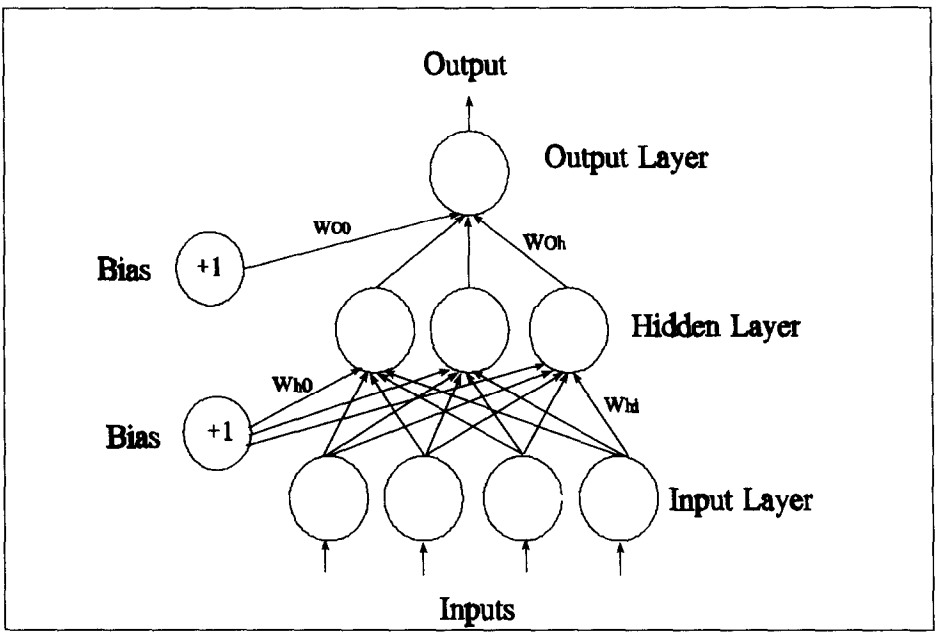


Fig. 1. A feedforward neural network.

BP networks are a class of feedforward neural networks with supervised learning rules. Feedforward refers to the direction of information flow from the input to the output layer. Inputs are passed through the neural network once to determine the output. Supervised learning is the process of comparing each of the network's forecasts with the known correct answer and adjusting the weights based on the resulting forecast error to minimize the error function.

The BP network is the most common multi-layer network estimated to be used in 80% of all applications [4] and is the focus of this paper since it is the most widely used in financial time series forecasting. Hornik et al. [11] showed that the standard BP network using an arbitrary transfer function can approximate any measurable function in a very precise and satisfactory manner, if a sufficient number of hidden neurons are used. Hecht-Nielsen [10] also demonstrated that a three-layer BP network can approximate any continuous mapping. Other neural networks less common in time series forecasting include recurrent networks, probablistic networks and fuzzy neural networks.

3. Steps in designing a neural network forecasting model

A method of designing a neural network forecasting model into distinct steps is used here. The eight-step design methodology presented in Table 2 draws on the steps outlined by Deboeck [5], Masters [18], Blum [3], and Nelson and Illingworth.

Table 2
Eight steps in designing a neural network forecasting model

Step 1: Variable selection
Step 2: Data collection
Step 3: Data preprocessing
Step 4: Training, testing, and validation sets
Step 5: Neural network paradigms
number of hidden layers
number of hidden neurons
number of output neurons
transfer functions
Step 6: Evaluation criteria
Step 7: Neural network training
number of training iterations
learning rate and momentum
Step 8: Implementation

The procedure is usually not a single-pass one, but may require visiting previous steps especially between training and variable selection.

3.1. Step 1: Variable selection

Success in designing a neural network depends on a clear understanding of the problem [21]. Knowing which input variables are important in the market being forecasted is critical. This is easier said than done because the very reason for relying on a neural network is for its powerful ability to detect complex nonlinear relationships among a number of different variables. However, economic theory can help in choosing variables which are likely important predictors. At this point in the design process, the concern is about the raw data from which a variety of indicators will be developed. These indicators will form the actual inputs to the neural network.

The financial researcher interested in forecasting market prices must decide whether to use both technical and fundamental economic inputs from one or more markets. Technical inputs are defined as lagged values of the dependent variable or indicators calculated from the lagged values. Fundamental inputs are economic variables which are believed to influence the dependent variable. The simplest neural network model uses lagged values of the dependent variable(s) or its first difference as inputs. Such models have outperformed traditional Box-Jenkins models in price forecasting, although not in all studies. [24,25]. A more popular approach is to calculate various technical indicators which are based only on past prices (and occasionally volume and/or open interest) of the market being forecasted [6]. As an additional improvement, intermarket data can be used since the close link between all kinds of markets, both domestically and internationally, suggests that using technical inputs from a number of interrelated markets should improve forecasting performance. For example, intermarket data such as the Deutsche mark/yen and pound cross rates and interest rate differentials could be

used as neural network inputs when forecasting the D-mark. Fundamental information such as the current account balance, money supply, or wholesale price index may also be helpful.

The frequency of the data depends on the objectives of the researcher. A typical off-floor trader in the stock or commodity futures markets would most likely use daily data if designing a neural network as a component of an overall trading system. An investor with a longer term horizon may use weekly or monthly data as inputs to the neural network to formulate the best asset mix rather than using a passive buy and hold strategy. An economist forecasting the GDP, unemployment, or other broad economic indicators would likely use monthly or quarterly data.

3.2. Step 2: Data collection

The researcher must consider cost and availability when collecting data for the variables chosen in the previous step. Technical data is readily available from many vendors at a reasonable cost whereas fundamental information is more difficult to obtain. Time spent collecting data cannot be used for preprocessing, training, and evaluating network performance. The vendor should have a reputation of providing high quality data; however, all data should still be checked for errors by examining day to day changes, ranges, logical consistency (e.g. high greater than or equal to close, open greater than or equal to low) and missing observations.

Missing observations which often exist, can be handled in a number of ways. All missing observations can be dropped or a second option is to assume that the missing observations remain the same by interpolating or averaging from nearby values. Dedicating an input neuron to the missing observations by coding it as a one if missing and zero otherwise is also often done.

When using fundamental data as an input in a neural network four issues must be kept in mind. First, the method of calculating the fundamental indicator should be consistent over the time series. Second, the data should not have been retroactively revised after its initial publication as is commonly done in databases since the revised numbers are not available in actual forecasting. Third, the data must be appropriately lagged as an input in the neural network since fundamental information is not available as quickly as market quotations. Fourth, the researcher should be confident that the source will continue to publish the particular fundamental information or other identical sources are available.

3.3. Step 3: Data preprocessing

Data preprocessing refers to analyzing and transforming the input and output variables to minimize noise, highlight important relationships, detect trends, and flatten the distribution of the variable to assist the neural network in learning the relevant patterns. Since neural networks are pattern matchers, the representation of the data is critical in designing a successful network. The input and output variables for which the data was collected are rarely fed into the network in raw form. At the very least, the raw data must be scaled between the upper and lower

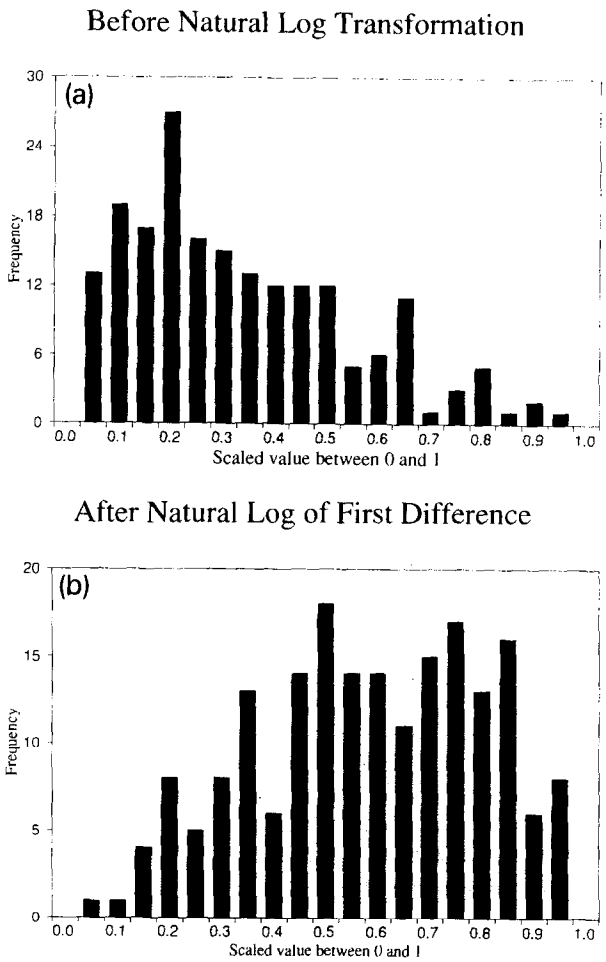


Fig. 2. Logarithmic transformation of monthly wheat futures trading volume at the Winnipeg Commodity Exchange, measured in 20 tonne job lots, 1977/78–1992/93.

bounds of the transfer functions (usually between zero and one or negative one and one).

Two of the most common data transformations in both traditional and neural network forecasting are first differencing and taking the natural log of a variable. First differencing, or using changes in a variable, can be used to remove a linear trend from the data. Logarithmic transformation is useful for data which can take on both small and large values and is characterized by an extended right hand tail distribution. The histograms shown in Fig. 2 illustrate the compressing effect of a logarithmic transformation on monthly futures trading volume for wheat. Logarithmic transformations also convert multiplicative or ratio relationships to additive which is believed to simplify and improve network training [18].

Another popular data transformation is to use ratios of input variables. Ratios highlight important relationships (e.g. hog/corn, financial statement ratios) while at the same time conserving degrees of freedom because fewer input neurons are required to code the independent variables.

Besides first differences, logs, and ratios, technical analysis can provide a neural network with a wealth of indicators including a variety of moving averages, oscillators, directional movement, and volatility filters. It is a good idea to use mix of different indicators to reduce variable redundancy and provide the network with the ability to adapt to changing market conditions through periodic retraining.

Smoothing both input and output data by using either simple or exponential moving averages is often employed. Empirical work on testing the efficient market hypothesis has found that prices exhibit time dependency or positive autocorrelation while price changes around a trend are somewhat random [26]. Therefore, attempting to predict price changes around the trend by using either unfiltered prices or price changes as inputs may prove to be difficult. Using moving averages to smooth the independent variables and forecasting trends may be a more promising approach.

Sampling or filtering of data refers to removing observations from the training and testing sets to create a more uniform distribution. The type of filtering employed should be consistent with the objectives of the researcher. For example, a histogram of price changes for a commodity would reveal many small changes from which an off-floor speculator cannot profit after deducting realistic execution costs. However, this dense region of the distribution will greatly impact the training of the neural network since small price changes account for the majority of the training facts. The network minimizes the sum of squared errors (or other error function) over all the training facts. By removing these small price changes, overall trading performance can be improved since the network specializes on the larger, potentially profitable price changes. It is possible for trading systems to be unprofitable even if the neural network predicted 85% of the turning points, as the turning points may be only small unimportant price changes [5]. On the other hand, a floor trader holding positions overnight is likely interested in these small price changes. The researcher must be clear on what exactly the neural network is supposed to learn. Another advantage of filtering is a decrease in the number of training facts which allows testing of more input variables, random starting weights, or hidden neurons rather than training large data sets.

In practice, data preprocessing involves much trial and error. One method to select appropriate input variables is to test various combinations. For example, a 'top 20' list of variables consisting of a variety of technical indicators could be pretested ten at a time with each combination differing by two or three variables. Although computationally intensive, this procedure recognizes the likelihood that some variables may be excellent predictors only when in combination with other variables. Chaos theory and statistical tests cannot make such a determination. Also, the top 20 list can be modified over time as the researcher gains experience on the type of preprocessing that works for his/her application. This approach is especially useful if the training set is small relative to the number of parameters

(weights) which is likely the case if all 20 input variables are presented to the neural network at once.

3.4. Step 4: Training, testing, and validation sets

Common practice is to divide the time series into three distinct sets called the training, testing, and validation (out-of-sample) sets. The training set is the largest set and is used by the neural network to learn the patterns present in the data. The testing set, ranging in size from 10% to 30% of the training set, is used to evaluate the generalization ability of a supposedly trained network. The researcher would select the network(s) which perform best on the testing set. A final check on the performance of the trained network is made using the validation set. The size of the validation set chosen must strike a balance between obtaining a sufficient sample size to evaluate a trained network and having enough remaining observations for both training and testing. The validation set should consist of the most recent contiguous observations. Care must be taken not to use the validation set as a testing set by repeatedly performing a series of train-test-validation steps and adjusting the input variables based on the network's performance on the validation set.

The testing set can be either randomly selected from the training set or consist of a set of observations immediately following the training set. The advantage of randomly selecting testing facts is that the danger of using a testing set characterized by one type of market is largely avoided. For example, a small testing set may only consist of prices in a strong uptrend. The testing set will favour networks which specialize on strong uptrends at the expense of networks which generalize by performing well on both uptrends and downtrends. The advantage of using the observations following the training set as testing facts is that these are the most recent observations (excluding the validation set) which may be more important than older data.

The randomly selected testing facts should not be replaced in the training set because this would bias the ability to evaluate generalization especially if the testing set is large relative to the training set (e.g. 30%). A deterministic method, such as selecting every nth observation as a testing fact, is also not recommended since it can result in cycles in the sampled data due solely to the sampling technique employed [18].

A more rigorous approach in evaluating neural networks is to use a walk-forward testing routine also known as either sliding or moving window testing. Popular in evaluating commodity trading systems, walk-forward testing involves dividing the data into a series of overlapping training-testing-validation sets. Each set is moved forward through the time series as shown in Fig. 3. Walk-forward testing attempts to simulate real-life trading and tests the robustness of the model through its frequent retraining on a large out-of-sample data set. In walk-forward testing, the size of the validation set drives the retraining frequency of the neural network. Frequent retraining is more time consuming, but allows the network to adapt more quickly to changing market conditions. The consistency or variation of

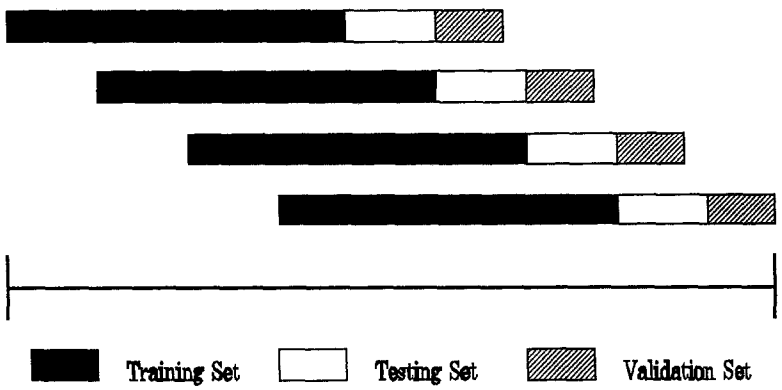


Fig. 3. Walk-forward sliding windows testing routine.

the results in the out-of-sample sets is an important performance measure. For example, in the case of commodity trading systems, a neural network with poor forecasting resulting in excessive equity drawdowns in any out-of-sample period would not be implemented in order to avoid risk of ruin.

It is recommended that the training and testing sets be scaled together since the purpose of the testing set is to determine the ability of the network to generalize. However, by no means, should the validation set be scaled with either the training or testing sets since this biases the integrity of the validation set as a final and independent check on the neural network. In actual use, the researcher has no way of knowing the exact range of future values, but only has a reasonable estimate based on the range of the training and/or testing sets.

3.5. Step 5: Neural network paradigms

There are an infinite number of ways to construct a neural network. Neurodynamics and architecture are two terms used to describe the way in which a neural network is organized. The combination of neurodynamics and architecture define the neural network's paradigm. Neurodynamics describe the properties of an individual neuron such as its transfer function and how the inputs are combined [21]. A neural network's architecture defines its structure including the number of neurons in each layer and the number and type of interconnections.

The number of input neurons is one of the easiest parameters to select once the independent variables have been preprocessed because each independent variable is represented by its own input neuron. This section will address the selection of the number of hidden layers, hidden layer neurons, and output neurons, and the transfer functions.

3.5.1. Number of hidden layers

The hidden layer(s) provide the network with its ability to generalize. In theory, a neural network with one hidden layer with a sufficient number of hidden

neurons is capable of approximating any continuous function. In practice, neural networks with one and occasionally two hidden layers are widely used and have performed very well. Increasing the number of hidden layers also increases computation time and the danger of overfitting which leads to poor out-of-sample forecasting performance. Overfitting occurs when a forecasting model has too few degrees of freedom. In other words, it has relatively few observations in relation to its parameters and therefore it is able to memorize individual points rather than learn the general patterns. In the case of neural networks, the number of weights, which is inexorably linked to the number of hidden layers and neurons, and the size of the training set (number of observations) determine the likelihood of overfitting [2,18]. The greater the number of weights relative to the size of the training set, the greater the ability of the network to memorize idiosyncrasies of individual observations. As a result, generalization for the validation set is lost and the model is of little use in actual forecasting.

Therefore, it is recommended that all neural networks should start with preferably one or at most two hidden layers. If a four-layer neural network (i.e. two hidden layers) proves unsatisfactory after having tested multiple hidden neurons using a reasonable number of randomly selected starting weights, then the researcher should modify the input variables a number of times before adding a third hidden layer. Both theory and virtually all empirical work to date suggests that networks with more than four layers will not improve the results.

3.5.2. Number of hidden neurons

Despite its importance, there is no 'magic' formula for selecting the optimum number of hidden neurons. Therefore, researchers fall back on experimentation. However, some rules of thumb have been advanced. A rough approximation can be obtained by the geometric pyramid rule proposed by Masters. For a three-layer network with n input neurons and m output neurons, the hidden layer would have $sqrt(n \times m)$ neurons. The actual number of hidden neurons can still range from one-half to two times the geometric pyramid rule value depending on the complexity of the problem. Baily and Thompson [1] suggest that the number of hidden neurons in a three-layer neural network should be 75% of the number of input neurons. Katz [13] indicates that an optimal number of hidden neurons will generally be found between one-half to three times the number of input neurons. Ersoy [8] proposes doubling the number of hidden neurons until the network's performance on the testing set deteriorates. Klimasauskas [15] suggests that there should be at least five times as many training facts as weights, which sets an upper limit on the number of input and hidden neurons.

It is important to note that the rules which calculate the number of hidden neurons as a multiple of the number of input neurons implicitly assume that the training set is at least twice as large as the number of weights and preferably four or more times as large. If this is not the case, then these rules of thumb can quickly lead to overfitted models since the number of hidden neurons is directly dependent on the number of input neurons (which in turn determine the number of weights). The solution is to either increase the size of the training set or, if this is

not possible, to set an upper limit on the number of input neurons so that the number of weights is at least half of the number of training facts. Selection of input variables becomes even more critical in such small networks since the luxury of the presenting the network with a large number of inputs and having it ignore the irrelevant ones has largely disappeared.

Selecting the 'best' number of hidden neurons involves experimentation. Three methods often used are the fixed, constructive, and destructive. In the fixed approach, a group of neural networks with different numbers of hidden neurons are trained and each is evaluated on the testing set using a reasonable number of randomly selected starting weights. The increment in the number of hidden neurons may be one, two, or more depending on the computational resources available. Plotting the evaluation criterion (e.g. sum of squared errors) on the testing set as a function of the number of hidden neurons for each neural network generally produces a bowl shaped error graph. The network with the least error found at the bottom of the bowl is selected because it is able to generalize best. This approach is time consuming, but generally works very well.

The constructive and destructive approaches involve changing the number of hidden neurons during training rather than creating separate networks each with a different number of hidden neurons, as in the fixed approach. Many commercial neural network software packages do not support the addition or removal of hidden neurons during training. The constructive approach involves adding hidden neurons until network performance starts deteriorating. The destructive approach is similar except that hidden neurons are removed during training.

Regardless of the method used to select the range of hidden neurons to be tested, the rule is to always select the network that performs best on the testing set with the least number of hidden neurons. When testing a range of hidden neurons it is important to keep all other parameters constant. Changing any parameter in effect creates a new neural network with a potentially different error surface which would needlessly complicate the selection of the optimum number of hidden neurons.

3.5.3. Number of output neurons

Deciding on the number of output neurons is somewhat more straightforward since there are compelling reasons to always use only one output neuron. Neural networks with multiple outputs, especially if these outputs are widely spaced, will produce inferior results as compared to a network with a single output [18]. A neural network trains by choosing weights such that the average error over all output neurons is minimized. For example, a neural network attempting to forecast one month ahead and six month ahead cattle futures prices will concentrate most of its effort on reducing the forecast with the largest error which is likely the six month forecast. As a result, a relatively large improvement in the one month forecast will not be made if it increases the absolute error of the six month forecasts by an amount greater than the absolute improvement of the one month forecast. The solution is to have the neural networks specialize by using separate networks for each forecast. Specialization also makes the trial and error design

procedure somewhat simpler since each neural network is smaller and fewer parameters need to be changed to fine tune the final model.

3.5.4. Transfer functions

Transfer functions are mathematical formulas that determine the output of a processing neuron. They are also referred to as transformation, squashing, activation, or threshold functions. The majority of current neural network models use the sigmoid (S-shaped) function, but others such as the hyperbolic tangent, step, ramping, arc tan, and linear have also been proposed. The purpose of the transfer function is to prevent outputs from reaching very large values which can 'paralyze' neural networks and thereby inhibit training.

Linear transfer functions are not useful for nonlinear mapping and classification. Levich and Thomas [16] and Kao and Ma [12] found that financial markets are nonlinear and have memory suggesting that nonlinear transfer functions are more appropriate. Transfer functions such as the sigmoid are commonly used for time series data because they are nonlinear and continuously differentiable which are desirable properties for network learning.

Klimasauskas [15] states that if the network is to learn average behaviour a sigmoid transfer function should be used while if learning involves deviations from the average, the hyperbolic tangent function works best. The ramping and step functions are recommended for binary variables since the sigmoid transfer function approaches zero and one asymptotically. In a standard BP network, the input layer neurons typically use linear transfer functions while all other neurons use a sigmoid function.

The raw data is usually scaled between 0 and 1 or -1 and $+1$, so it is consistent with the type of transfer function which is being used. Linear and mean/standard deviation scaling are two of the most common methods used in neural networks. In linear scaling all observations are linearly scaled between the minimum and maximum values according to the following formula:

$$SV = TF_{\min} + (TF_{\max} - TF_{\min}) \times \frac{(D - D_{\min})}{(D_{\max} - D_{\min})} \tag{1}$$

where SV is the scaled value, $TF_{\min}$ and $TF_{\max}$ are the respective minimum and maximum values of the transfer function, D is the value of the observation, and $D_{\min}$ and $D_{\max}$ are the respective minimum and maximum values of all observations.

Simple linear scaling is susceptible to outliers because it does not change the uniformity of the distribution, but merely scales it into the appropriate range for the transfer function. In the S&P 500 data presented in Fig. 4, linear scaling results in 98.6% of the training facts being contained within 10% of the neuron's activation range. Proper training is unlikely to take place with such a distribution. In mean and standard deviation scaling all values plus or minus x number of standard deviations from the mean are mapped to one and zero, respectively. All other values are linearly mapped between zero and one. This type of scaling creates a more uniform distribution and is more appropriate for data which has

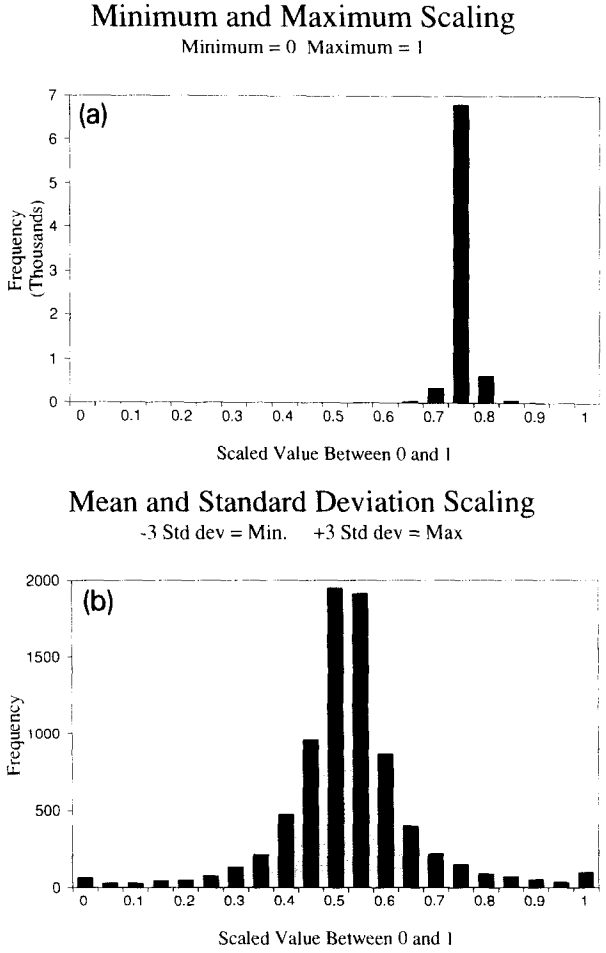


Fig. 4. Two common scaling methods applied to daily cash S&P 500 closing prices, 1962–1993.

not been sampled in any way. Most neural network software programs will automatically scale all variables into the appropriate range. However, it is always a good idea to look at histograms of the scaled input and output variables.

3.6. Step 6: Evaluation criteria

The most common error function minimized in neural networks is the sum of squared errors. Other error functions offered by software vendors include least absolute deviations, least fourth powers, asymmetric least squares, and percentage differences. These error functions may not be the final evaluation criteria since other common forecasting evaluation methods such as the mean absolute percentage error (MAPE) are typically not minimized in neural networks.

In the case of commodity trading systems, the neural network forecasts would be converted into buy/sell signals according to a predetermined criterion. For example, all forecasts greater than 0.8 or 0.9 can be considered buy signals and all forecasts less than 0.2 or 0.1 as sell signals [9]. The buy/sell signals are then fed into a program to calculate some type of risk adjusted return and the networks with the best risk adjusted return (not the lowest testing set error) would be selected. Low forecast errors and trading profits are not necessarily synonymous since a single large trade forecasted incorrectly by the neural network could have accounted for most of the trading system's profits.

Filtering the time series to remove many of the smaller price changes can largely prevent the situation where a neural network with high turning point forecasting accuracy remains unprofitable. Also, the value of any single trading system can only be established within the context of the user's portfolio of commodity systems. In this regard, neural networks may be especially useful if they behave more like counter trend systems as opposed to the more common trend following systems used by commodity funds.

3.7. Step 7: Neural network training

Training a neural network to learn patterns in the data involves iteratively presenting it with examples of the correct known answers. The objective of training is to find the set of weights between the neurons that determine the global minimum of the error function. Unless the model is overfitted, this set of weights should provide good generalization. The BP network uses a gradient descent training algorithm which adjusts the weights to move down the steepest slope of the error surface. Finding the global minimum is not guaranteed since the error surface can include many local minima in which the algorithm can become 'stuck'. A momentum term and five to ten random sets of starting weights can improve the chances of reaching a global minimum. This section will discuss when to stop training a neural network and the selection of learning rate and momentum values.

3.7.1. Number of training iterations

There are two schools of thought regarding the point at which training should be stopped. The first stresses the danger of getting trapped in a local minimum and the difficulty of reaching a global minimum. The researcher should only stop training until there is no improvement in the error function based on a reasonable number of randomly selected starting weights [18]. The point at which the network does not improve is called convergence. The second view advocates a series of train-test interruptions [5,20]. Training is stopped after a predetermined number of iterations and the network's ability to generalize on the testing set is evaluated and training is resumed. Generalization is the idea that a model based on a sample of the data is suitable for forecasting the general population. The network for which the testing set error bottoms out is chosen since it is assumed to generalize best.

The criticism of the train-test procedure is that additional train-test interruptions could cause the error on the testing set to fall further before rising again or it

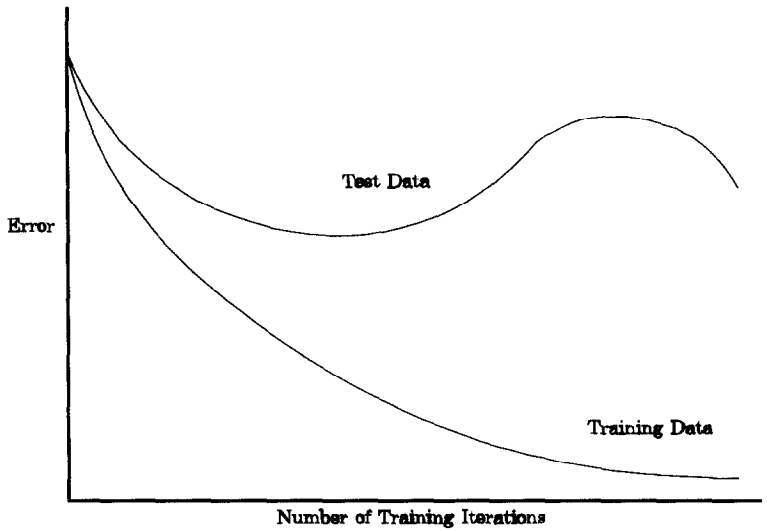


Fig. 5. Possible neural network training and testing set errors.

could even fall asymptotically (Fig. 5). In other words, the researcher has no way of knowing if additional training could improve the generalization ability of the network especially since starting weights are randomized.

Both schools of thought agree that generalization on the validation set is the ultimate goal and both use testing sets to evaluate a large number of networks. The point at which the two approaches depart centres on the notion of overtraining versus overfitting. The convergence approach states that there is no such thing as overtraining only overfitting. Overfitting is simply a symptom of a network that has too many weights. The solution is to reduce the number of hidden neurons (or hidden layers if there is more than one) and/or increase the size of the training set. The train-test approach attempts to guard against overfitting by stopping training based on the ability of the network to generalize.

The advantage of the convergence approach is that one can be more confident that the global minimum was reached. Replication is likely more difficult for the train-test approach given that starting weights are usually randomized and the mean correlation can fluctuate wildly as training proceeds. Another advantage is that the researcher has two less parameters to worry about; namely the point at which to stop training and the method to evaluate which of the train-test networks is optimal. An advantage of the train-test approach may be that networks with few degrees of freedom can be implemented with better generalization than convergence training which would result in overfitting. However, empirical work has not specifically addressed this issue. The train-test approach also requires less training time.

The objective of convergence training is to reach a global minimum. This requires training for a sufficient number of iterations using a reasonable number of

randomly selected starting weights. Even then there is no guarantee with a BP network that a global minimum is reached since it may become trapped in a local minimum. In practice, computational resources are limited and tradeoffs arise. The researcher must juggle the number of input variable combinations to be trained, the interval of hidden neurons over which each network is to be tested, the number of randomly selected starting weights, and the maximum number of runs. For example, 50 input variable combinations tested over three different hidden neurons with five sets of randomly selected starting weights and a maximum number of runs of 4,000 result in 3,000,000 iterations (epochs). The same computational time is required for ten input variable combinations tested over six hidden neurons with ten sets of randomly selected starting weights and 5,000 epochs.

One method to determine a reasonable value for the maximum number of runs is to plot the mean correlation, sum of squared errors, or other appropriate error measure for each iteration or at predetermined intervals up to the point where improvement is negligible (usually up to a maximum of 10,000 iterations). Each iteration can be easily plotted if the neural network software creates a statistics file or, if this is not the case, the mean correlation can be recorded at intervals of 100 or 200 from the computer monitor. After plotting the mean correlation for a number of randomly selected starting weights, the researcher can choose the maximum number of runs based on the point where the mean correlation stops increasing quickly and flattens.

Many studies that mention the number of training iterations report convergence from 85 to 5,000 iterations [6,14]. However, the range is very wide as 50,000 and 191,400 iterations [15,22] and training times of 60 hours have also been reported [9]. Training is affected by many parameters such as the choice of learning rate and momentum values, proprietary improvements to the BP algorithm, among others, which differ between studies and so it is difficult to determine a general value for the maximum number of runs. Also, the numerical precision of the neural network software can affect training because the slope of the error derivative can become very small causing some neural network programs to move in the wrong direction due to round off errors which can quickly build up in the highly iterative training algorithm. It recommended that researchers determine the number of iterations required to achieve negligible improvement for their particular problem and test as many randomly selected starting weights as computational constraints allow.

3.7.2. Learning rate and momentum

A BP network is trained using a gradient descent algorithm which follows the contours of the error surface by always moving down the steepest slope. The objective of training is to minimize the total squared errors, defined as follows [19]:

$$E = \tfrac{1}{2}\sum_{h}^{M} E_h = \tfrac{1}{2}\sum_{h}^{M}\sum_{i}^{N}(t_{hi} - O_{hi})^2 \tag{2}$$

where E is the total error of all patterns, E_h represents the error on pattern h, the index h ranges over the set of input patterns, and i refers to the ith output

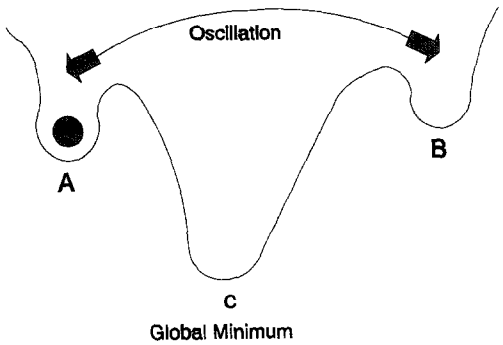


Fig. 6. Simplified graphical representation of a neural network error surface.

neuron. The variable t_{hi} is the desired output for the ith output neuron when the hth pattern is presented, and O_{hi} is the actual output of the ith output neuron when pattern h is presented. The learning rule to adjust the weight between neuron i and j is defined as:

$$\delta_{hi} = (t_{hi} - O_{hi})O_{hi}(1 - O_{hi}) \tag{3}$$

$$\delta_{hi} = O_{hi}(1 - O_{hi})\sum_{k}^{N}\delta_{hk}w_{jk} \tag{4}$$

$$\Delta w_{ij}(n + 1) = \varepsilon(\delta_{hi}O_{hj}) \tag{5}$$

where n is the presentation number, δ_{hi} is the error signal of neuron i for pattern h, and ε is the learning rate. The learning rate is a constant of proportionality which determines the size of the weight changes. The weight change of a neuron is proportional to the impact of the weight from that neuron on the error. The error signal for an output neuron and a hidden neuron are calculated by Eq. (3) and (4), respectively.

As an analogy to the BP training algorithm, one can consider the problem of trying to throw a ball from point A to point C in Fig. 6, although in reality the error surface is multidimensional and cannot be represented in a graphical format. The force on the ball is analogous to the learning rate. Applying too much force will cause the ball to overshoot its target and it may never return to point A or it can oscillate between points A and B. During training, a learning rate that is too high is revealed when the error function is changing wildly without showing a continued improvement. Too little force on the ball and it will be unable to escape from point A which is evident during training when there is very little or no improvement in the error function. A very small learning rate also requires more training time. In either case, the researcher must adjust the learning rate during training or 'brainwash' the network by randomizing all weights and changing the learning rate for the new run through the training set.

One method to increase the learning rate and thereby speed up training time without leading to oscillation is to include a momentum term in the backpropaga-

tion learning rule. The momentum term determines how past weight changes affect current weight changes. The modified BP training rule is defined as follows:

$$\Delta w_{ij}(n+l) = \varepsilon(\delta_{hi} O_{hj}) + \alpha \Delta w_{ij}(n) \tag{6}$$

where α is the momentum term, and all other terms as previously defined.

The momentum term suppresses side to side oscillations by filtering out high-frequency variations. Each new search direction is a weighted sum of the current and the previous gradients. Such a two-period moving average of gradients filters out rapid fluctuations in the learning rate. Momentum values that are too great will prevent the algorithm from following the twists and turns in weight space. McClelland and Rumelhart [19] indicate that the momentum term is especially useful in error spaces containing long ravines that are characterized by steep, high walls and a gently sloping floor. Without a momentum term, a very small learning rate would be required to move down the floor of the ravine which would require excessive training time. By dampening the oscillations between the ravine walls, the momentum term can allow a higher learning rate to be used.

Most neural network software programs provide default values for learning rate and momentum that typically work well. Initial learning rates used in previous work vary widely from 0.1 to 0.9 [31,23]. Common practice is to start training with a higher learning rate such as 0.7 and decrease as training proceeds. Many neural network programs will automatically decrease the learning rate and increase momentum values as convergence is reached.

3.8. Step 8: Implementation

The implementation step is listed as the last one, but in fact requires careful consideration prior to collecting data. Data availability, evaluation criteria, and training times are all shaped by the environment in which the neural network will be deployed. Most neural network software vendors provide the means by which trained networks can be implemented either in the neural network program itself or as an executable file. If not, a trained network can be easily created in a spreadsheet by knowing its architecture, transfer functions, and weights. Care should be taken that all data transformations, scaling, and other parameters remain the same from testing to actual use.

An advantage of neural networks is their ability to adapt to changing market conditions through periodic retraining. Once deployed, a neural network's performance will degrade over time unless retraining takes place. However, even with periodic retraining, there is no guarantee that network performance can be maintained as the independent variables selected may have become less important.

It is recommended that the frequency of retraining for the deployed network should be the same as used during testing on the final model. However, when testing a large number of networks to obtain the final model, less frequent retraining is acceptable in order to keep training times reasonable. A good model should be robust with respect to retraining frequency and will usually improve as retraining takes place more often.

4. Summary

Neural networks are a type of artificial intelligence technology that mimic the human brain's powerful ability to recognize patterns. The application of neural networks in the field of financial economics for such tasks as pattern recognition, classification, and forecasting is relatively new [28]. In theory, neural networks are capable of approximating any continuous function. Such flexibility provides a potentially powerful forecasting tool, while at the same time, the large number of parameters that must be selected complicates the design process. In practice, designing a neural network forecasting model involves much trial and error.

Therefore, the objective of this paper was to provide a practical, non-technical introduction to designing a neural network forecasting model using economic time series data. The design procedure was divided into eight steps; (1) variable selection, (2) data collection, (3) data preprocessing, (4) training, testing and validation sets, (5) neural network paradigms, (6) evaluation criteria, (7) neural network training, and (8) implementation. In each step, some rules of thumb, common mistakes, parameter selection, and points of disagreement among neural network developers were discussed.

The success of neural network applications for an individual researcher depends on three key factors. First, the researcher must have the time, patience and resources to experiment. The relatively recent application of neural networks in financial economics and the large number parameters means that only broad rules of thumb have been published and much experimentation and imagination is in order. Second, the neural network software must allow automated routines such as walk-forward testing, optimization of hidden neurons, and testing of input variable combinations; either through direct programming or the use of batch/script files. Many programs available are inadequate for serious development work and will quickly frustrate even the most determined researcher. Third, the researcher must maintain a good set of records that list all parameters for each network tested since any parameter listed in Table 1 may turn out to cause a significant change in neural network performance. In this way, a library of what is successful and what is not is built up.

Acknowledgement

The authors would like to thank the reviewers for their helpful comments, and the Winnipeg Commodity Exchange Fellowship Program for funding this research.

References

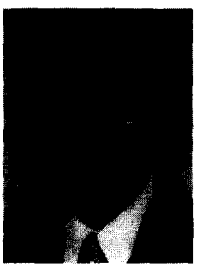
[1] D. Baily and D.M. Thompson, Developing neural network applications, *AI Expert*, (Sep. (1990) 33–41.

[2] E.B. Baum and D. Haussler, What size net gives valid generalization?, *Neural Computat.* 6 (1989) 151–160.

[3] A. Blum, *Neural Networks in C + + : An Object-Oriented Framework for Building Connectionist Systems* (Wiley, New York, 1992).
[4] M. Caudill, The view from now. *AI Expert*, (June 1992) 24–31.
[5] G.J. Deboeck, Ed. *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, (Wiley, New York, 1994).
[6] G.J. Deboeck and M. Cader, Trading U.S. treasury notes with a portfolio of neural net models, in G.J. Deboeck, Ed. *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, (Wiley, New York, 1994) 102–122.
[7] J. Egan, Artificially intelligent investing, *U.S. News & World Report*, 56 (March 1993) 73.
[8] O. Ersoy, Tutorial at Hawaii International Conference on Systems Sciences, Jan. 1990.
[9] L. Hamm, B. Wade Brorsen and R. Sharda, Futures trading with a neural network *NCR-134 Conf. on Applied Commodity Analysis, Price Forecasting, and Market Risk Management Proc.* Chicago (1993) 286–296.
[10] R. Hecht-Nielsen, *Neurocomputing*, (Addison Wesley, Menlo Park, CA, 1989).
[11] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators; *Neural Networks* 2 (1989) 359–366.
[12] G.W. Kao and C.K. Ma, Memories, heteroscedasticity and prices limit in currency futures markets; *J. Futures Markets* 12 (1992) 672–692.
[13] J.O. Katz, Developing neural network forecasters for trading, *Technical Analysis of Stocks and Commodities* April (1992) 58–70.
[14] K.L. Klaussen and J.W. Uhrig, Cash soybean price prediction with neural networks; *NCR-134 Conf. on Applied Commodity Analysis, Price Forecasting, and Market Risk Management Proc.*, Chicago (1994) 56–65.
[15] C.C. Klimasauskas, Applying neural networks, in R.R. Trippi and E. Turban, eds., *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance* (Probus, Chicago, 1993) 64–65.
[16] R.M. Levich and L.R. Thomas, The significance of technical trading rule profits in the foreign exchange market: A bootstrap approach; in *Strategic Currency Investing - Trading and Hedging in the Foreign Exchange Market*, (Probus, Chicago, 1993) 336–365.
[17] R.P. Lippman, An introduction to computing with neural nets, *IEEE ASSP Mag.* (April 1987) 4–22.
[18] T. Masters, *Practical Neural Network Recipes in C + +*, (Academic Press, New York, 1993).
[19] J.L. McClelland, D.E. Rumelhart and the PDP Group. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, (Cambridge, MA, MIT Press, 1986).
[20] L. Mendelsohn, Training neural networks, *Technical Analysis of Stocks and Commodities*, (Nov. 1993) 40–48.
[21] M. McCord Nelson, W.T. Illingworth, *A Practical to Guide to Neural Nets*, (Addison Wesley, Reading, MA, 1991).
[22] M.D. Odom and R. Sharda, A neural network model for bankruptcy prediction, *Proc. IEEE Int. Conf. on Neural Networks*, San Diego (1992) II163–II168.
[23] L.M. Salchenberger, E. Mine Cinar and N. A. Lash, Neural networks: A new tool for predicting thrift failures; *Decision Sciences* 23 (July/Aug. 1992) 899–916.
[24] R. Sharda and R.B. Patil, A connectionist approach to time series prediction: An empirical test; in G.J. Deboeck, Ed., *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, (Wiley, New York, 1994) 451–464.
[25] Z. Tang, C. de Almedia and P.A. Fishwick, Time series forecasting using neural networks vs. Box-Jenkins (Methodology; *International Workshop on Neural Networks*, Auburn, AL (Jan. 1990).
[26] W.G. Tomek and S.F. Querin, Random processes in prices and technical analysis; *J. Futures Markets* 4 (1984) 15–23.
[27] R.R. Trippi and D. DeSieno Trading equity index futures with a neural network; *J Portfolio Management* 19 (1992) 27–33.
[28] R.R. Trippi and E. Turban, eds. *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real-World Performance*, (Probus, Chicago, 1993).

[29] H. White, Learning in neural networks: A statistical perspective, *Neural Computat.* 4 (1989) 425–464.
[30] H. White, A.R. Gallant, K. Hornik, M. Stinchcombe and J. Wooldridge, *Artificial Neural Networks: Approximation and Learning Theory*, (Blackwell, Cambridge, MA, 1992.
[31] Y. Yoon and G. Swales, Predicting stock price performance: A neural network approach, *Proc. 24th Annual Int. Con. of Systems Sciences*, Chicago (1991) 156–162.

Iebeling Kaastra is Manager-Pricing at the Canadian Wheat Board in Winnipeg, Canada where his work includes hedging using derivatives markets. He was born in Europe and later moved to Canada. He obtained both a Bachelor's degree in Finance and a Master's degree in Agricultural Economics from the University of Manitoba in Canada. His Masters thesis research involved forecasting futures trading volume for various grains traded on the Winnipeg Commodity Exchange. He has written in a number of academic journals and trade magazines, including *Journal of Futures Markets* and *Grainews*.



Milton Boyd is an Associate Professor at the University of Manitoba in Winnipeg, Canada, where he teaches and researches in the area of commodity markets and price analysis. Milton grew up in Canada and then obtained a Bachelors degree in Finance from Seattle Pacific University and a Doctorate degree from Purdue University in the USA. Most recently his research has been on using computer artificial intelligence for price prediction and commodity trading.

He has written over one hundred articles on commodity markets and price analysis for business and academics. He has written in a number of trade magazines, including ***Grainews*** where he is a columnist and contributing editor. As well, he teaches and coordinates University short courses on futures and options in collaboration with the Winnipeg Commodity Exchange. He is also a co-author of ***Hedging Canadian Grains***, published by the Winnipeg Commodity Exchange, and has lectured in a number of countries around the world.