AN ARTIFICIAL WALK DOWN WALL STREET:

Can Intraday Stock Returns Be Predicted Using Artificial Neural Networks?

Jens Olve Bøvre Peder Kristian Viervoll

Norwegian School of Economics and Business Administration (NHH)

June 2009, Bergen

Advisor: Associate Professor Jonas Andersson

“If a man will begin with certainties, he shall end in doubts, but if he will be content to begin with doubts, he shall end in certainties.”

Sir Francis Bacon

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

FIGURES

TABLES

TABLES IN APPENDIX A

ABSTRACT

Financial markets are complex evolved dynamic systems. Due to its irregularity, financial time series forecasting is regarded as a rather challenging task. In recent years, artificial neural network applications in finance, for such tasks as pattern recognition, classification, and time series forecasting have dramatically increased. The objective of this paper is to present this powerful framework and attempt to use it to predict the stock return series of four publicly listed companies on the New York Stock Exchange. Our findings coincide with those of Burton Malkiel in his book, *A Random Walk down Wall Street*; no conclusive evidence is found that our proposed models can predict the stock return series better than a random walk.

ACKNOWLEDGEMENTS

First of all, we would like to thank our advisor, Associate Professor Jonas Andersson at NHH, for his guidance on theoretical reasoning, instructive comments, support and patience. Special gratitude is extended to Professor Terje Kristensen at the Bergen College University for his invaluable insight in the field of neural networks and artificial intelligence. Your comments and suggestions were highly appreciated. We would also like to thank Professor Jostein Lillestøl for his help and discussion on miscellaneous topics. Finally we would like to thank PhD candidate at NHH, Knut Nygaard, for helpful comments on market microstructure.

1 INTRODUCTION

Forecasting and detecting trends and patterns in financial data are of great interest to the business world because the more precise the forecasts are the more utility is likely to be gained from acting on them. Stock market prediction is an area which attracts a great deal of attention since once the prediction of returns is successful, monetary rewards will be substantial.

To achieve this, a powerful framework that can generalize the underlying process of stock returns must be defined, applied and evaluated. In financial forecasting of returns, a classic workhorse framework is the *autoregressive integrated moving average* ($ARIMA$) process and variations of it. An $ARIMA$ process is linear in its nature, yet there is practically no evidence that the underlying process of stock returns is completely linear (Mills, 1990).

This leads us to believe that nonlinear frameworks may provide more reliable predictions. The first objective of this paper is to present a nonlinear model framework popularly called the *artificial neural network* (ANN). The novelty of ANNs lies in their ability to model both linear and nonlinear processes without a priori assumptions about the nature of the generating process. Examples of such assumptions are normality of residuals and collinearity among explanatory variables.

ANNs are mathematical models inspired by the human nervous system. They have been successfully applied to a broad spectrum of low-level cognitive tasks such as pattern recognition of cancer cells (Moallemi, 1991), voice recognition (Kuah et al., 1994), inspection for defects in a production process (Seiji et al., 2004) and the use of hyphenation in written language (Kristensen et al., 1997). When it comes to economics and finance, ANNs have been used in areas like portfolio management (Fernandez and Gomez, 2007) and credit rating (Atiya, 2001) in addition to regular financial time series forecasting.

One of the first researchers to actually use an ANN on financial time series was White (1988). White applied a simple ANN on a time series of the IBM stock price with the aim to prove the efficient market hypothesis. He did not find evidence

against it and concluded that a *random walk* was still the best model for stock price movements[1]. On the other hand, studies like Bosarge (1993) and Refenes et al. (1995) introduced more advanced ANNs that challenged the efficient market hypothesis. (See Hill et al., 1996 for a comprehensive survey on this topic).

ANNs are often classified as *black boxes*. A Black box is defined as a device or theoretical construct with known or specified performance characteristics but unknown or unspecified constituents and means of operation. This leads us to the second objective of this paper, which is to show the reader that ANNs share similarities to multiple regression models and, in some cases, are a continuation of the $ARIMA$ framework. The last objective of this paper is to use our proposed models in an attempt to forecast the stock return time series on an intraday basis on four companies listed on the New York Stock Exchange (NYSE).

The structure of this paper is as follows. We start by presenting theory on some classic time series forecast methods which will be used as a comparison to our ANN models. We will then show and explain the typical structure of an ANN and how to build a simple neural network. Subsequently we discuss the data foundation plus the filtering and sorting process of the raw data obtained for the analysis. Next, we discuss the modeling method together with calibration of the various models including our own enhanced version of the initial ANN. The penultimate chapter deals with the actual empirical analysis on four selected stock return time series and the results. Finally, we summarize our findings and make some recommendations for future research.

[1] The random walk hypothesis is a financial theory stating that stock market prices evolve according to a random walk and thus the prices of the stock market cannot be predicted. The term was popularized by the 1973 book, *A Random Walk Down Wall Street*, by Burton Malkiel, currently a Professor of Economics and Finance at Princeton University.

2 METHODOLOGY

In this section we present an overview of relevant theory which will form the base of our proposed models in Section 4. This includes an introduction to ANNs and a parametric framework used as comparison to our proposed models; the $ARIMA$ framework. We aim to provide the reader with an intuitive explanation of the concepts behind the models[2].

2.1 THE ARIMA FRAMEWORK

The autoregressive integrated moving average ($ARIMA$) model which is a parametric model most commonly used to predict future points in a time series was first introduced by Box and Jenkins (1970). The model combines the generalized AR model and the generalized MA model with a differencing factor. Consider the time series:

$$(1) \qquad y_1, y_2, \dots, y_{t-1}, y_t$$

Consider also the occurrence of random shocks ε at each time step:

$$(2) \qquad \varepsilon_1, \varepsilon_2, \dots, \varepsilon_{t-1}, \varepsilon_t$$

$\{\varepsilon_k\}$ is Gaussian white noise and is by definition *independent and identically distributed* (i.i.d.). For supplementary explanations and proof on the subjects in the remaining part of section 2.1, any book on financial time series analysis, like Tsay (2002), will do.

2.1.1 THE AUTOREGRESSIVE PROCESS

If $\{y_t\}$ is generated by

$$(3) \qquad y_t = \phi_0 + \phi_1 y_{t-1} + \varepsilon_t$$

[2] References to appendices or external sources for elaborate derivations and proof not essential to our analysis are given where necessary.

we call it an $AR(1)$ process. This is simply a linear regression with explanatory variable y_{t-1} and coefficients ϕ_0 and ϕ_1 as parameters. The generalized AR model for p lags is given by:

$$(4) \qquad y_t = \phi_0 + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t = \phi_0 + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t$$

where the parameter p is a non-negative integer and indicates that the preceding p values in the time series serve as explanatory variables for y_t and $\phi_i (i = 1,2, \dots, p)$ as parameters.

2.1.2 THE MOVING AVERAGE PROCESS

A moving average process of order 1 is defined by:

$$(5) \qquad y_t = \theta_0 + \varepsilon_t - \theta_1 \varepsilon_{t-1}$$

This is abbreviated $MA(1)$ and can be generalized for order q:

$$(6) \qquad y_t = \theta_0 + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q} = \theta_0 - \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

The variable y_t is according to the $MA(q)$ model a weighted average of ε_t to ε_{t-q} with $\theta_j (j = 1,2, \dots, q)$ as parameters to estimate.

2.1.3 THE AUTOREGRESSIVE MOVING AVERAGE PROCESS

The autoregressive moving average process combines the two processes above and is abbreviated $ARMA(p, q)$:

$$(7) \qquad y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

2.1.4 THE AUTOREGRESSIVE INTEGRATED MOVING AVERAGE PROCESS

The time series y_t is said to be an $ARIMA(p, d, q)$ process if the difference series y_t follows a stationary and invertible $ARMA(p, q)$ process:

$$(8) \qquad \Delta^d y_t = \sum_{i=1}^{p} \phi_i \Delta^d y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t \quad \text{where} \quad \Delta^d y_t = y_t - y_{t-d}$$

The autoregressive part of the process, $ARIMA(p, 0,0)$, refers to the importance of previous values in the time series. The value at time y_{t-k} may have an effect on the value at time y_{t-k+1} and y_{t-k+2}. As time passes, the effect will decrease exponentially towards zero. The differentiation part of the process, $ARIMA(0, d, 0)$, removes trend and drift in the time series and makes non-stationary data stationary. The final part of the process, $ARIMA(0,0, q)$, relates the previous q shocks with the present shock ε_t. The estimation of coefficients and the forecast method are discussed in Section 4.

2.1.5 RANDOM WALK

A random walk is a mathematical formulation of a trajectory of successive random steps. A generalized $ARIMA(p, d, q)$ process with no autoregressive part ($p = 0$), difference factor of one ($d = 1$) and no moving average part ($q = 0$) is the definition of a random walk:

$$(9) \qquad y_t = y_{t-1} + \varepsilon_t$$

The model states that the next value in the time series is only dependent on the previous observation plus a shock variable which is i.i.d.

2.2 SELECTING ARIMA MODELS

2.2.1 AUTOCORRELATION FUNCTION

The autocorrelation function (ACF) describes the general linear dependence between y_t and its past values y_{t-h}. The correlation coefficient between y_t and y_{t-h} is called the lag h autocorrelation and will be denoted ρ_h. If we have a *weakly stationary* time series, i.e. a series that is constant for characteristics such as expectation and variance, ρ_h is defined by:

$$(10) \qquad \rho_h = \frac{\mathrm{Cov}(y_t, y_{t-h})}{\sqrt{\mathrm{Var}(y_t)Var(y_{t-h})}} = \frac{\mathrm{Cov}(y_t, y_{t-h})}{\mathrm{Var}(y_t)}$$

because of the property $Var(y_t) = Var(y_{t-h})$ of the weakly stationary time series. We therefore obtain that $\rho_0 = 1$, $\rho_h = \rho_{-h}$ and $-1 \leq \rho_h < 1$. Also, a weakly stationary series y_t, is not serially correlated if and only if $\rho_h = 0$ for all $h > 0$. For a given sample of a weakly stationary series $\{y_t\}_{t=1}^{T}$, let $\overline{y}$ be the sample mean $\overline{y}$=$\sum_{t=1}^{T} y_t$ /T. The lag-h sample autocorrelation of y_t is defined as

$$(11) \qquad \hat{\rho}_h = \frac{\sum_{t=h+1}^{T}(y_t - \overline{y})(y_{t-h} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2} \quad \text{where} \quad 0 \leq h < T-1$$

If y_t is an i.i.d. sequence satisfying $\mathrm{E}(y_t^2) < \infty$, then $\hat{\rho}_h$ is asymptotically normal with zero mean and variance $1/T$ for any fixed positive integer h. In finite samples $\hat{\rho}_h$ is a biased estimator of ρ_h, but in most financial applications the bias is not so serious because of the relatively large size of T.

2.2.2 PARTIAL AUTOCORRELATION FUNCTION

The partial autocorrelation function ($PACF$) adjusts for the impact the intervening lags $y_{t-1}, y_{t-2}, \ldots, y_{t-h+1}$ have on the correlation between y_t and y_{t-h}. The hth partial autocorrelation is the coefficient ϕ_{hh} in the linear model

$$(12) \qquad y_t = \phi_{h1} y_{t-1} + \phi_{h2} y_{t-2} + \cdots + \phi_{hh} y_{t-h} + \varepsilon_t$$

and determine the additional correlation between y_t and y_{t-h} after adjustments have been made for the intervening lags.

2.2.3 IDENTIFYING THE ORDER OF AR AND MA MODELS

When identifying p and q of an $ARMA(p, q)$ the autocorrelation plot and the partial autocorrelation plot are usually the primary tools. The sample autocorrelation plot and the sample partial autocorrelation plot are assessed to their respectively theoretical behavior. For a stationary $AR(1)$ process, the sample autocorrelation function is exponentially decreasing or a damped sinusoidal. For $AR(p)$ processes

with $p > 1$ the sample ACF is usually a mixture of exponentially decreasing and damped sinusoidal components. For higher-order autoregressive processes, the sample autocorrelation needs to be supplemented with a partial autocorrelation plot. The partial autocorrelation of an $AR(p)$ process becomes zero at lag $\geq p + 1$, we therefore check the sample partial autocorrelation function to search for evidence that it is significantly different from zero.

The autocorrelation function of a $MA(q)$ process becomes zero at lag $\geq q + 1$, it is thus necessary to check the sample autocorrelation function to see where it in effect becomes zero. Table 1 provides a summary for some of the properties of the ACF and the $PACF$ for different $ARMA$ processes.

TABLE 1 – PROPERTIES OF ACF AND PACF

PROCESS	ACF	PCF
White noise	All $\rho_h = 0$ $(h \neq 0)$	All $\phi_{hh} = 0$
$AR(1)$: $a_1 > 0$	Direct exponential decay: $\rho_h = {a_h}^1$	$\phi_{11} = \rho_1;\ \phi_{hh} = 0\ for\ h \geq 2$
$AR(1)$: $a_1 < 0$	Oscillating decay: $\rho_h = {a_h}^1$	$\phi_{11} = \rho_1;\ \phi_{hh} = 0\ for\ h \geq 2$
$AR(p)$	Decays toward zero, coefficients may oscillate	Spikes through lag p. All $\phi_{hh} = 0\ for\ h > p$
$MA(1)$: $\beta > 0$	Positive spike at lag 1. $\rho_h = 0\ for\ h \geq 2$	Osc. decay: $\phi_{11} > 0$
$MA(1)$: $\beta < 0$	Negative spike at lag 1. $\rho_h = 0\ for\ h \geq 2$	Geom. decay: $\phi_{11} < 0$
$ARMA(1,1)$: $a_1 > 0$	Exp. decay beginning at lag 1. Sign ρ_1 = sign($a_h + \beta$)	Osc. decay beginning at lag 1. $\phi_{11} = \rho_1$
$ARMA(1,1)$: $a_1 < 0$	Osc. decay beginning at lag 1. Sign ρ_1 = sign($a_h + \beta$)	Exp. decay beginning at lag 1. $\phi_{11} = \rho_1$
$ARMA(p,q)$	Decay (direct or osc.) beginning at lag q	Decay (direct or osc.) beginning at lag p

COMMENT: $ARMA$ model selection table found in Enders (2005). Specific patterns in ACF and PACF can be used for classification of the underlying process in a time series.

2.2.4 THE AKAIKE AND BAYESIAN INFORMATION CRITERION

Identifying p and q can be difficult in practice. It is not given that ACF and $PACF$ will give reasonable results. However, there have been developed different criteria that try to meet the shortcomings of the abovementioned theoretical functions. Two of these are the Akaike and the Bayesian Information Criteria commonly abbreviated AIC and BIC. They are both based on the maximum likelihood estimation approach. The AIC can be said to describe the tradeoff between bias and variance (i.e. precision and complexity) of the model (Akaike, 1974). The AIC is defined as:

$$(13) \qquad AIC = 2\ln(L) - 2n$$

Where L is the maximized likelihood function of the fitted model and n is the number of parameters in the model. When estimating model parameters using maximum likelihood estimation, it is possible to increase the likelihood by adding additional parameters, which may result in overfitting[3]. The BIC tries to resolve the same problem by introducing a different penalty term for the number of parameters in the model (Schwarz, 1978). BIC is defined as:

$$(14) \qquad BIC = 2\ln(L) - \ln(n)\, n$$

Also here L is the maximized likelihood function of the fitted model and n is the number of parameters in the model. The AIC and BIC are not used for hypothesis testing, rather it is a tool for model selection. Given a data set, several competing models may be ranked according to these criteria. The model receiving the lowest value is considered to be better. From the values yielded by (13) and (14) one may infer that the top three models are better than the rest of the models. One should not assign a numerical value above which a given model is rejected. We will use both criteria when we select our models derived from the $ARIMA$ framework in Section 4.

[3] In statistics, overfitting refers to a statistical model that has too many parameters. An absurd and false model may fit perfectly if the model has enough complexity by comparison to the amount of data available. When the degrees of freedom in parameter selection exceed the information content of the data, this leads to arbitrariness in the final (fitted) model parameters which reduces or destroys the ability of the model to generalize beyond the fitting data.

2.3 ARTIFICIAL NEURAL NETWORKS

An *artificial neural network,* simply *neural network* or ANN is a model inspired by the human nervous system. McCulloch and Pitts (1943) and Rosenblatt (1958) are examples of some of the earliest research on neural networks and focuses on the simulation of the human brain. The human nervous system is a network of billions of neurons (nerve cells) in the human body. Each neuron is connected to a number of other neurons forming a very complex parallel system which interconnect and communicate by sending chemical signals between each other. Depending on the strength of each signal, a neuron relays a signal if the aggregated signal from other neurons is strong enough. An ANN can be considered to emulate a *learning entity* that acquire dependencies from its environment and act accordingly. In this setting, however, we abstract from the biological sphere and will define ANNs as pure mathematical models.

The literature on ANNs is vast and there exists a wide range of different network structures each with their own advantages and disadvantages. For a good overview and expansion on the theory presented this section, see for example Rumelhart and McClelland (1986).The most common network structure is the *multilayer perceptron* (MLP). Although a MLP network has a specific functional form it is more flexible than traditional linear models. The main advantage is that MLP networks can approximate any nonlinear continuous function using one *input layer*, one *hidden layer* and one *output layer* with a sufficient number of *nodes* (Hornik et al. 1989). In the following sections we will review *architectures* and the *learning process* for a generic MLP network and finish with a paragraph concerning the black box criticism directed against neural networks in general.

2.3.1 NETWORK PROPERTIES

A typical structure of a MLP network is presented in Figure 1 below. The network consists of an input layer, a hidden layer and an output layer. The input layer consists of input nodes (similar to neurons in the human nervous system) which feed the network with relevant data. This data form a linear combination (weighted sum) and is sent to the hidden layer for *activation*. The weighted sum in each hidden node is

activated using an *activation function* that transforms the linear combination to fit into a pre-determined interval. The transformed sum is then weighted and sent to output nodes in the output layer.

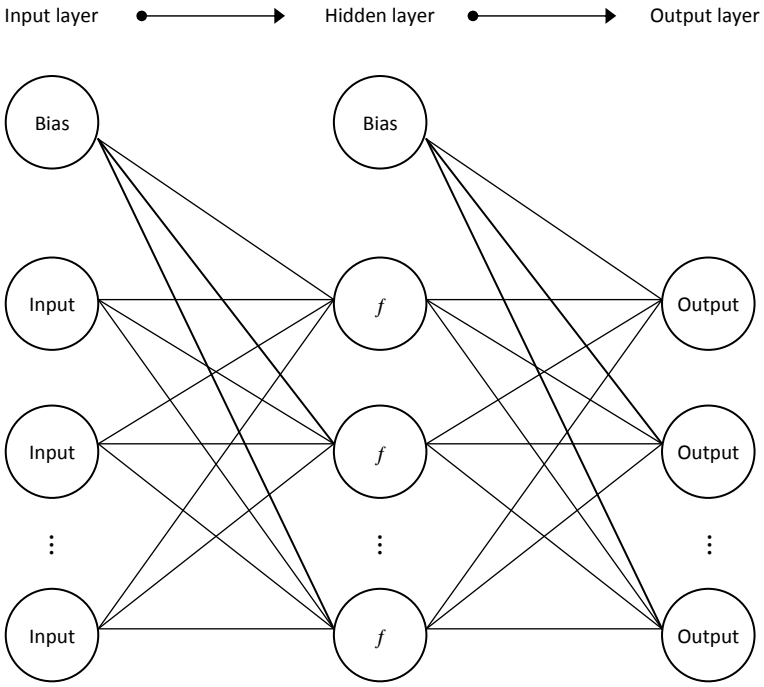


FIGURE 1 – NEURAL NETWORK STRUCTURE I: Graphical interpretation of a general three layered perceptron. The arrows show the way information flows through the network. The input values are transferred (weighted) through a function, f, in the hidden layer and subsequently transferred (weighted) to the output layer as the model output.

Input nodes

The input nodes are naturally the part where values are admitted into the model. For a time series this would typically be lagged values of y_t or other characteristic measures. Input nodes are equivalent to independent variables using statistical terminology. The input layer can then be handled as a vector of input variables:

$$A = [a_1 \quad a_2 \quad \cdots \quad a_I] \tag{15}$$

Connection weights

Connections between nodes are actually weights that determine the relevance of the transfer signal from a node to another. These weights can be compared to the statistical term *parameters* in frameworks like $ARIMA$ and regression models. ANNs are therefore sometimes referred to as both non-parametric and semi-parametric

models. Weights are typically randomized at the beginning of the *training period* and then adjusted using appropriate methods during training. The weights between an input layer and a hidden layer can be presented using matrix terminology. Consider an $I \times J$ matrix:

$$W^A = \begin{bmatrix} \phi_{11}^A & \phi_{12}^A & \cdots & \phi_{1J}^A \\ \phi_{21}^A & \phi_{22}^A & \cdots & \phi_{2J}^A \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{I1}^A & \phi_{I2}^A & \cdots & \phi_{IJ}^A \end{bmatrix} \quad (16)$$

where I is the number of input nodes and J is the number of hidden nodes. In addition to the input nodes, there is a bias node which serves as a constant in the model and its value is equal to 1 at the beginning of the training period. The bias changes along with the weights throughout the training and corresponds to the constant coefficient in a standard regression. The bias weights are denoted ϕ_{0j}^A. Weights between the hidden layer and the output layer can be presented as a matrix using the same terminology as above. Consider a $J \times K$ matrix:

$$W^B = \begin{bmatrix} \phi_{11}^B & \phi_{12}^B & \cdots & \phi_{1K}^B \\ \phi_{21}^B & \phi_{22}^B & \cdots & \phi_{2K}^B \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{J1}^B & \phi_{J2}^B & \cdots & \phi_{JK}^B \end{bmatrix} \quad (17)$$

where J is the number of hidden nodes and K is the number of output nodes. Like the input layer, the hidden layer also has bias weights, ϕ_{0k}^B.

Activation function

The activation function is located in each hidden node (and output node if necessary) and is normally of sigmoid type. A sigmoid function produces an s-shaped curve and is real-valued and differentiable, having either a non-negative or non-positive first derivative and exactly one inflection point[4]. The function has two asymptotes when $x \to \pm\infty$. The logistic function produces a value inside the interval $[0,1]$. If the values used as input or desired output is in another region it would be more reasonable to use another activation function. An example of this is the hyperbolic tangent, which

[4] Point on the curve where the second derivative changes sign and goes from concave to convex (or vice versa).

produces values inside the interval $[-1,1]$. The network's training algorithm makes use of the first derivative of the activation function. Because of the trivial deduction of their first derivative, the two functions mentioned above are the most commonly used activation functions in neural network literature (McNelis, 2005).

The logistic function and its first derivative:

$$(18) \qquad f(x) = \frac{1}{(1+e^{-x})} \qquad f'(x) = f(x)\big(1 - f(x)\big)$$

The hyperbolic tangent and its first derivative:

$$(19) \qquad f(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad f'(x) = 1 - \big(f(x)\big)^2$$

The two functions are graphed below with x-values ranging from -5 to $+5$. The s-shaped curves reflects their appeal where small changes in an already low or high variable have very little effect.

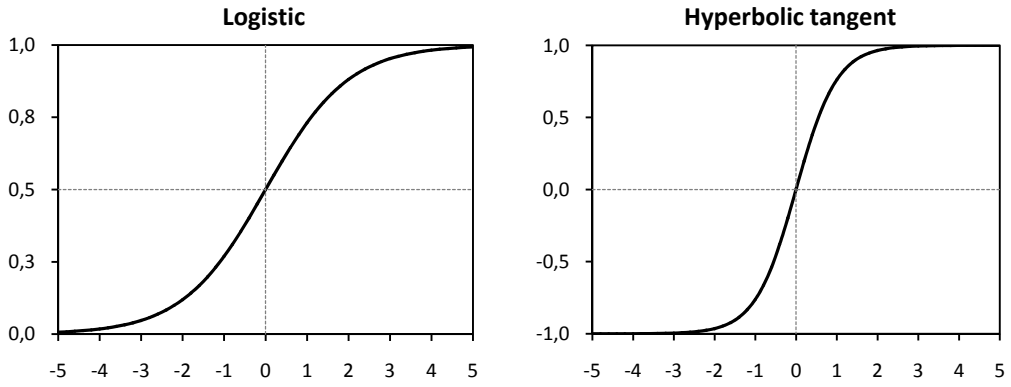


FIGURE 2 – SIGMOID ACTIVATION FUNCTIONS: Logistic sigmoid function (left) and the hyperbolic tangent (right) for $x\epsilon[-5,5]$. This feature is often preferred in economic theory. Using a simple example: a small change in an already very low or very high NOK/USD FX rate will not have a significant effect on the decision to import or export goods. However, the same change when the exchange rate lay somewhere between the more extreme values would create a more pronounced impact on import and export demand.

Many neural network models use hard limit threshold functions that produce two or more values, depending on the node sum. An example of this is the binary function with a limit threshold τ. The function $f(x)$ would yield 1 for $x > \tau$ and 0 for $x \leq \tau$:

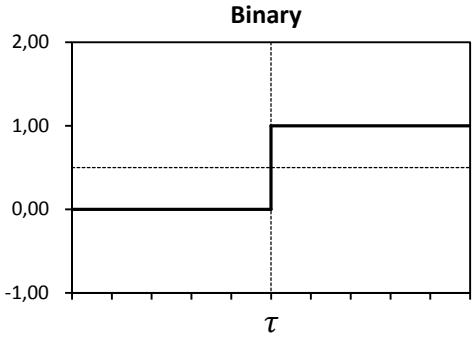


FIGURE 3 – THRESHOLD ACTIVATION FUNCTION: Graph showing the mapping of a threshold function that produces either zero or unity depending on the threshold limit τ.

Hidden nodes

The hidden nodes receive a weighted sum from the different input nodes plus the input bias which is put through the activation function. The hidden layer as a vector of hidden nodes is defined as:

$$(20) \qquad B = [b_1 \quad b_2 \quad \cdots \quad b_J] \text{ where } b_j = f\left(\phi_{0j}^A + \sum_{i=1}^{I} \phi_{ij}^A a_i\right) \text{ and } j = 1,2,\dots,J$$

Output nodes

The output node is the final destination for the transferred data. The output may be linear in the sense that the weighted sum from all the hidden nodes plus the bias term is considered as the final output. Alternatively one can use an activation function in the output node to produce a value inside a desired interval. Output nodes are analogous to dependent variables using statistical terminology. The output value for output node k is:

$$(21) \qquad c_k = f\left(\phi_{0k}^B + \sum_{j=1}^{J} \phi_{jk}^B b_j\right) \quad \text{where} \quad k = 1,2,\dots,K$$

Substituting for b_j forms the following model framework with a activation function in the output node:

$$(22) \qquad c_k = f\left(\phi_{0k}^B + \sum_{j=1}^{J} \phi_{jk}^B f\left(\phi_{0j}^A + \sum_{i=1}^{I} \phi_{ij}^A a_i\right)\right) \quad \text{where} \quad k = 1,2,\dots,K$$

For time series prediction, using one output node, lagged values of y_t and, assuming residuals are additive white noise, the functional form would be:

$$(23) \qquad y_t = f\left(\phi_{01}^B + \sum_{j=1}^{J} \phi_{j1}^B f\left(\phi_{0j}^A + \sum_{i=1}^{I} \phi_{ij}^A y_{t-i}\right)\right) + \varepsilon_t$$

Figure 4 is a continuation of Figure 1 using the general notation introduced above. As we can see, the network inhabits strong parallel mapping skills. Acknowledging that, we also see ANNs biggest disadvantage; complexity increases exponentially with network size. One of the most important issues when building an ANN is to find a balance between precision and complexity (cf. Section 2.2.4).

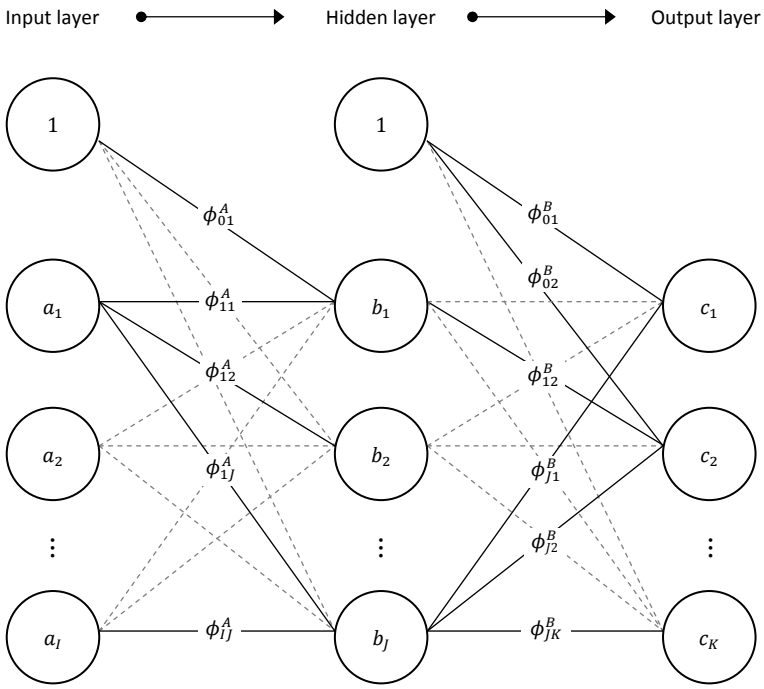


FIGURE 4 – NEURAL NETWORK STRUCTURE II: The general architecture for a three layered multilayer perceptron showing the weights between specific nodes and layers. The reason for dashed grey lines is to make it easier to see some of the specific weights between nodes and layers. This is a fully connected MLP with one hidden layer. For more complex structures, some of the weights between nodes can be deleted and there can be several hidden layers for added complexity.

2.3.2 LEARNING ALGORITHM

When the architecture is set, the MLP network weights must be estimated. This process is called *training* in the neural network literature. There are several methods available for training, some more complex than others. The most popular learning paradigm is the *gradient descent algorithm*, also called *back-propagation*. This concept was popularized by Rumelhart and McClelland (1986). The mathematical structure of the back-propagation algorithm is quite trivial compared to more advanced learning algorithms. The objective of the training algorithm is to minimize the mean square error (MSE) of the entire training set of data which is defined as:

$$(24) \qquad E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} e_k^n \qquad \text{where} \qquad e_k^n = \frac{1}{2}(c_k^* - c_k)^2$$

where E is the total error of all patterns presented to the model and k refers to the output node. c_k is the actual output from the model and c_k^* is the desired output (what the model should have forecasted). e_k^n is the instantaneous error resulting from the difference between c_k and c_k^* in output node k in training pattern n. This error is propagated backwards through the network to allocate it to the right weights and adjust them accordingly.

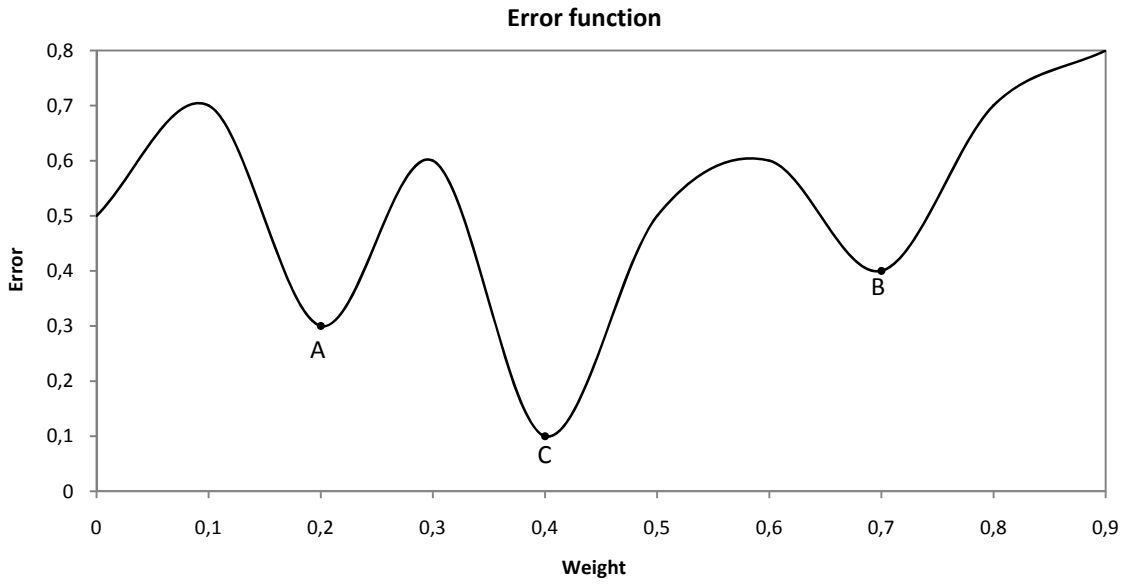


FIGURE 5 – ERROR FUNCTION EXAMPLE: Simplified graphical representation of a neural network error surface. Point A and B are local minimums of the error function E. Point C is the desirable result in this conventional example.

The weight changes are made by implementing the following equations where i, j and k refers to the input, hidden and output layer:

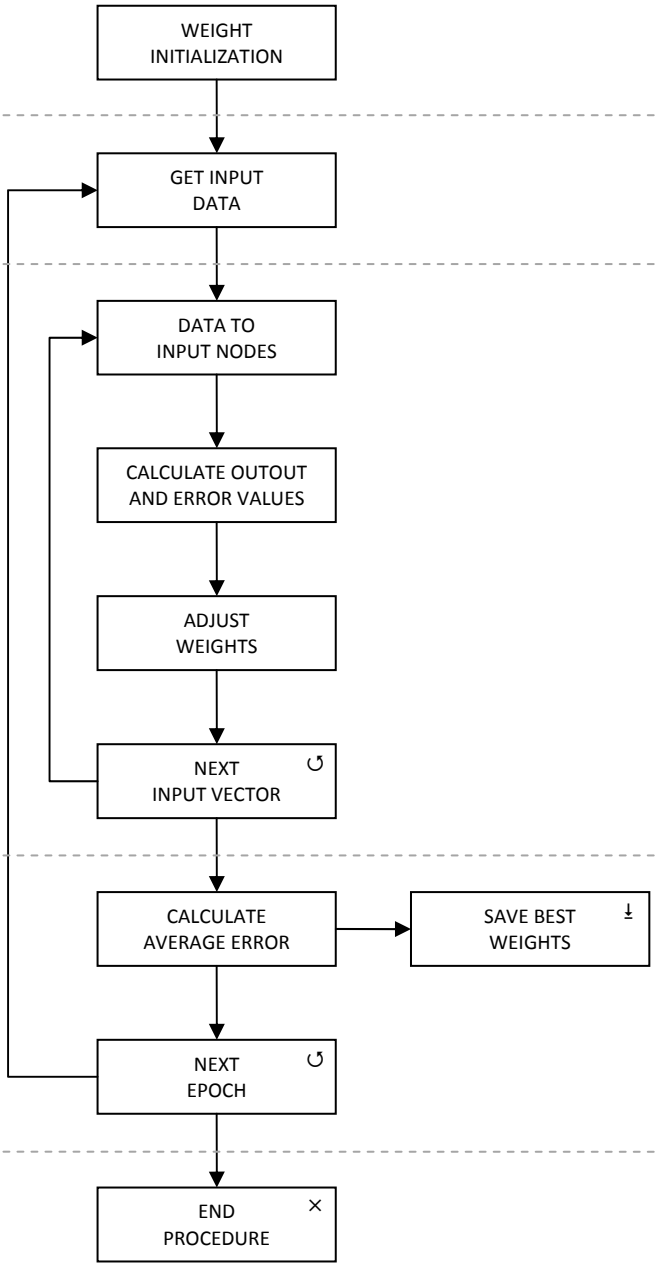
$$(25) \qquad \Delta_t \phi_{jk}^B = \eta \delta_k s_k + \gamma \Delta_{t-1} \phi_{jk}^B \quad \text{where} \quad \delta_k = (c_k^* - c_k) f'(s_k)$$

$$(26) \qquad \Delta_t \phi_{ij}^A = \eta \delta_j s_j + \gamma \Delta_{t-1} \phi_{ij}^A \quad \text{where} \quad \delta_j = \left(\sum_{k=1}^{K} \delta_k \phi_{jk}^B \right) f'(s_j)$$

Where s_j and s_k are the values *before* they are put through the activation function. The derivations of equations (25) and (26) are presented in Appendix B. The δ_{kj} and δ_j terms are the local *gradients* for the hidden layer and input layer. Gradients represent a sensitivity factor, determining the direction of the search in weight space for the optimal weight. η is called the *learning rate* and γ is called *momentum*, both with values between zero and unity. The smaller we set η, the smaller the weight change is from one iteration to the next. The momentum term γ is added to increase the learning rate without destabilizing the network.

This procedure is used on every weight in the network and iterated through the entire training set of data. A full iteration of the N training patterns is called an *epoch*. After an epoch, the average squared error and weights are saved if the error is the lowest yet or if any stopping criteria have been met. Such criteria could be an upper limit of epochs or total time elapsed. Training the network usually requires many epochs so that we avoid ending up with weights that place us in the vicinity of a local minimum of the error function. The weight adjustment equation for weights between the hidden and the output layer (25) is slightly different than the weight change for weighs between the input layer and the hidden layer (26), because the error is dependent on subsequent weights in the network.

To follow the different steps, it may be easier to follow the learning algorithm looking at a schematic presentation. It is also more convenient for programming purposes because the flow chart below can easily be adapted to any programming language:



STEP 1
Select random weights for connections between all layers.

STEP 2
Feed network with the first input vector of N vectors in the training set.

STEP 3
Let the network calculate the output values for each output node. The corresponding error measure is consequently propagated backwards in the network.

Using equation (24) and (25), the algorithm adjusts weights so that the error in the output nodes is reduced.

The error from each output node is aggregated and saved. When this is done, the next input vector in the training data set is fed to the network. This procedure is repeated until all the training vectors are run through and a total aggregated error saved.

STEP 4
When the algorithm has run trough the entire training set, it calculates the average error and save the network weights if the average error is the lowest yet.

This run is called an epoch and the algorithm run through a given set of epochs before it stops.

STEP 5
The procedure stops when it reaches the maximum number of epochs or any other early-stopping measure.

LEGEND:

× End of the algorithm. This is achieved either with a successful run or by an error

↺ A loop. This would correspond to a *do procedure while a condition is met* routine

↧ Writing a result either in an array or any other format chosen by the programmer

FIGURE 6 – FLOWCHART OF LEARNING ALGORITHM: Diagram of the back-propagation algorithm. This process is easily programmable in any desired language using loops and logical parameters. Comments on the code are presented in Appendix C.

The rationale behind back-propagation could also be easier to comprehend using an analogy: suppose you throw a bouncing ball from a mountain top and down into a valley. The ball will bounce down the mountainside (error surface) eventually stopping on a plateau. This plateau can either be the bottom of the valley or some flat rock face somewhere on the way down. Let η be the measure of how bouncy the ball is and γ the weight of the ball. If the ball is too bouncy (high η) it may reach the bottom very quickly, but in an unstable manner bounce away and disappear. If the weight of the ball is too high (high γ), the ball may settle on a plateau in the mountainside. The outcome of optimal bounce and weight would be that the ball settles at the bottom of the valley.

If the model could see the error function like the conventional example in Figure 5, then it would have chosen point C. This would be the global minimum of the error function E. Because the error function is a hyper plane in a multi-dimensional space, the model cannot see the entire function but only the gradient in a specific point. As a consequence it can end up in point A or B. If the algorithm reaches such a saddle point (i.e. a plateau on the mountainside) it may converge to this solution. With the momentum term, the training algorithm is capable of jumping out of local minima points. The precision of these jumps depends on tuning the momentum term. A low momentum term can lead to oscillation and unstable behavior.

2.3.3 BLACK BOX

Neural networks have been considered by many to be black boxes. It is hardly the mathematics behind the learning algorithm which is the problem. The weights, however, are not easily interpreted. If we decide to have zero hidden layers and linear activation functions, (22) will be simplified to:

$$(27) \qquad y_t = \phi_{01}^A + \sum_{i=1}^{I} \phi_i^A y_{t-i} + \varepsilon_t$$

This is a normal multiple regression, and more specifically, equivalent to the autoregressive process defined by (4). Here, every ϕ has an interpretable meaning. Adding layers simply introduce a dynamic property to the regression. Appending one

hidden layer and keeping the activation functions linear we derive the following dynamic multiple regression:

$$(28) \qquad y_t = \phi_{01}^B + \sum_{j=1}^{J} \phi_{j1}^B \left(\phi_{0j}^A + \sum_{i=1}^{I} \phi_{ij}^A y_{t-i} \right) + \varepsilon_t$$

This MLP network is more powerful than the AR process due to the nonlinear functional mapping from past observations to the future value and can be thought of as a semi-parametric regression model (Chan et al., 2007).

This shows us that a neural network is not a magicians hat, but merely an advanced regression model which given the right input and structure may be able to give a reasonable result. This requires an understanding of network structure, how the network performs calculations and how the output data should be analyzed.

2.4 MEASURING FORECAST ACCURACY

Clearly, one cannot determine whether a forecasting model is good or not based upon a single forecast and one realization. Many econometric forecasting studies evaluate the models success using statistical loss functions, like the previously mentioned MSE:

$$(29) \qquad MSE = \frac{1}{T} \sum_{t=1}^{T} \left(y_{t+h} - \hat{y}_{t,h} \right)^2$$

where $\hat{y}_{t,h}$ is the h-step-ahead forecast and y_{t+h} is the actual value. This provides a quadratic loss function rendering all values positive. It makes sense when dealing with stock returns which could cancel each other out if we where to use real values.

Another criterion which can be used is the U-statistic (Theil, 1966):

$$(30) \qquad U = \frac{\sqrt{\sum_{t=1}^{T} \left(\frac{y_{t+h} - \hat{y}_{t,h}}{y_{t+h}} \right)^2}}{\sqrt{\sum_{t=1}^{T} \left(\frac{y_{t+h} - \hat{z}_{t,h}}{y_{t+h}} \right)^2}}$$

where $\hat{z}_{t,h}$ is the forecast from a simple model such as a random walk. A U-statistic of one implies that the model under consideration and the benchmark model are equally accurate (or inaccurate). For $U < 1$ implies that the model in question is superior to the benchmark and vice versa for $U > 1$. Albeit useful, the U-statistic is not without problems. If the simple model equals the actual value at time t, the statistic will be infinite (because of a zero denominator).

It is not necessarily the case that models are useful in practical situations even if they are classified as accurate. For example, Gerlow et al. (1993) show that the accuracy of forecasts according to traditional statistical criteria may give little guide to the potential profitability of employing those forecasts in a market trading strategy.

On the other hand, models that can accurately forecast the sign of future returns have been found to be more profitable (Leitch and Tanner, 1991). They suggest the following indicator to show if a model inhabits the ability to predict direction changes regardless of their magnitude:

$$(31) \qquad S = \frac{1}{T}\sum_{t=1}^{T} s_{t+h} \quad \text{where} \quad s_{t+h} = \begin{cases} 1 \text{ if } \left(y_{t+h}\hat{y}_{t,h}\right) > 0 \\ 0 \text{ else} \end{cases}$$

Consider the following case: a simple model (e.g. RW) predicts the next point in a stock return time series to be -1 percent. A developed model predicts 3 percent. The actual value is 1 percent. MSE would equally rank the two models. Conversely, the S-statistic would rank the developed model better than the RW because it predicts in the right direction. We will utilize all three measures when we assess the prediction abilities of the models.

3 DATA

The data we have used in our empirical analysis are obtained from the Trade and Quote (TAQ) database. The TAQ database is a collection of intraday trades and quotes for all securities listed on the New York Stock Exchange (NYSE), the American Stock Exchange (AMEX) and the Nasdaq National Market System (NASDAQ). TAQ provides historical tick by tick data reported on consolidated tapes of operation[5]. Aside from programming and modeling, the most time consuming aspect of this analysis were preprocessing of data. Even though TAQ provides excellent background material for empirical analysis, the data is far from usable, at least from a statistical perspective, without proper filtering and sorting. The last part of this chapter is dedicated to explain what we have done with the data used as input in our models. This is done because of the lack of data presentation in empirical papers on similar subjects. We found that most research on financial time series forecasting using neural network methodology does not mention how the data was processed before it was used in the proposed model. Preprocessing is time consuming and tedious and were automated using sorting algorithms. Information about this algorithm is presented in Appendix C. Before we discuss the actual data set, we make use of the following section for a brief introduction to high frequency data and market microstructure, two topics which are important to our analysis.

3.1 HIGH FREQUENCY DATA AND MARKET MICROSTRUCTURE

Market microstructure is the functional setup of a market. This specific field in economics deals with the process of how exchanges occur in the market, like the process by which the price for an asset is determined. In some markets, like the stock market, prices are a result of a negotiation[6]. The seller asks for a certain price while the buyer presents his bid. The actual trade of an asset at any point in time will be the result of equilibrium between selling and buying agents. Because of the spread

[5] Combined ticker tapes of the NYSE and AMEX used for listed equity securities. Tape A covers the NYSE-listed securities and is used to identify the originating market. Tape B does the same for AMEX-listed securities and also reports on securities listed on regional stock exchanges.

[6] Other markets may have different price processes, like auctioning (e.g. Sotheby's) or they may simply be posted by the seller (e.g. the local supermarket).

between these bid and ask quotes, the actual trade price will bounce between these prices. This phenomenon is called the *bid-ask bounce*. When the frequency of recorded transactions increases, this bounce will feed noise to the time series, and must be handled accordingly. A more thorough introduction to market microstructure is not necessary for the sake of this analysis but can be found in for example O'Hara (1995).

When performing an econometric analysis, the choice of data frequency is important. Some frequencies are better suited for various analyses than others. In theory, increasing the data frequency on stock data will in the limit produce a perfect estimate of the variance, σ^2. In practice, higher frequency data may contain unwanted noise. This is the case with high frequency stock data. Even if stock data is sampled in the highest possible frequency, an estimate of σ^2 would not be entirely accurate due to microstructure noise (Aït-Sahalia, 1996). Figure 7 below show the stock price movement for Exxon Mobile Corporation (XOM) in monthly, weekly, daily and intraday (five minute intervals) over a four month period.

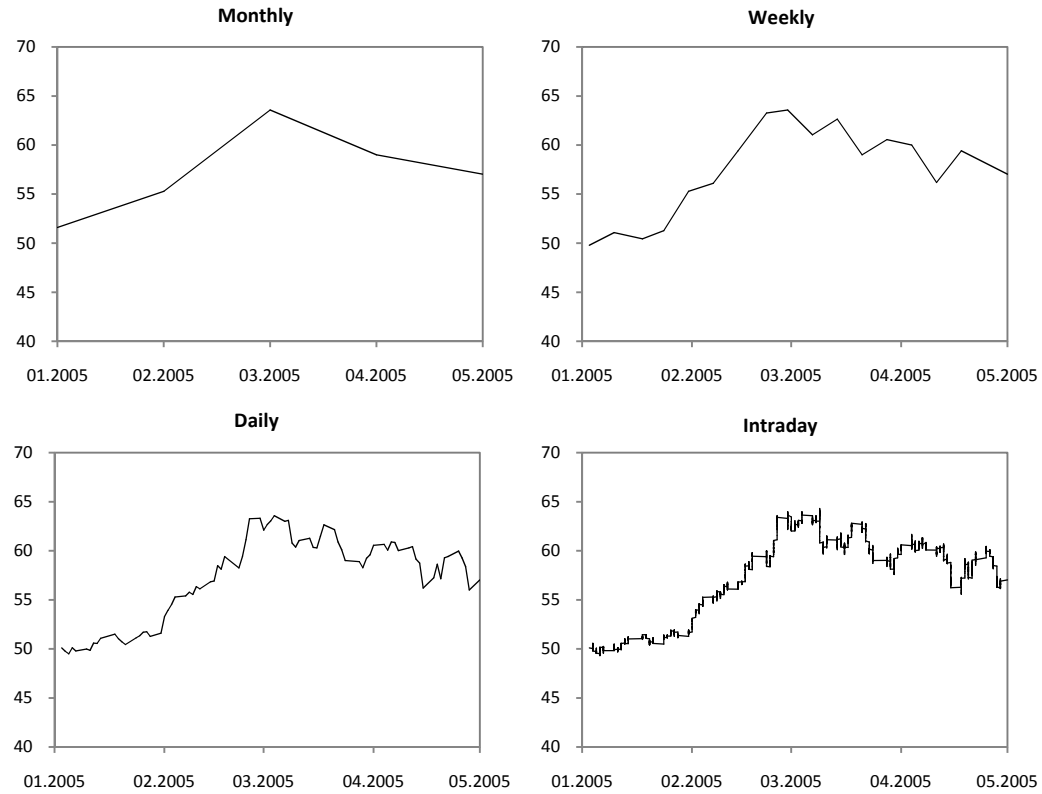


FIGURE 7 – DATA FREQUENCY CHARTS: Different intervals on the same data set. Clockwise from upper left; monthly, weekly, daily and five minute intervals of the XOM stock price series. All data are from the TAQ database.

For a portfolio analyst at a passively managed mutual fund the monthly data would most likely be a good choice for analyses. Weekly or daily data would probably be interesting for an actively managed mutual fund (or hedge fund) and intraday data for a day trader. We face the same problem; chose an optimal frequency with the goal to minimize noise, which will corrupt empirical studies if present.

The data collected from the TAQ database is the highest possible frequency obtainable, where every trade is recorded. The duration between trades fluctuates and as a consequence, the data is irregularly spaced. This imposes problems because most econometric models are specified for fixed intervals[7]. We need to aggregate trades and quotes so that econometric modeling can be performed. Aggregating may remove some of the noise in the time series, but doing so may also cause more damage. Jordà and Marcellino (2002) suggest that temporal aggregation of high-frequency, irregularly spaced data *'can generate non-normality, conditional heteroskedasticity, and leptokurtosis'*. Aït-Sahalia and Mykland (2003) propose a statistical approach to estimate an optimal interval for intraday data. We have not pursued this approach. Instead we chose an interval of five minutes which gives us 78 fixed intervals per day. This is a sensible starting point based on articles by Omrane and Oppens (2005) and Bollerslev et al. (2006). Furthermore, we deliberately choose stocks which are highly liquid in the sense that they are traded regularly. This assures that relevant data is present in every five minute interval.

3.2 DATA SET

We tried to select a period from the last decade where the equity market and the economy in general would be considered to be in a normal state. This excludes the period leading up to and following the 1997 Asian financial crisis, the period leading up to and following the Dot-Com bubble that climaxed in the early 2000s and the post-2007 global financial crisis. Our data cover the period from January 3, 2005 to January 10, 2005. This equals 612 observations, which will be split in three parts; an estimation set containing the first 372 observations, a validation set with the

[7] However, some econometric models for data observed at irregular time intervals exists. See for example Engle and Russell (1998) or Broersen and Bos (2006) for estimating time series with irregularly spaced data.

following 180 observations and finally a prediction set made up of the last 60 observations. Looking at the time frame of our predictions we feel that this amount of data is sufficient. We presume that information older than the horizon we have selected have little or no impact on the stock price because it should, in theory, be reflected in today's price.

3.2.1 STOCKS

We perform the analysis on four NYSE stocks from four different industries: Exxon Mobil (XOM), Proctor & Gamble (PG), Goldman Sachs (GS) and Frontline (FRO). These are highly liquid stocks with little duration between trades.

3.2.2 TRADE PRICE AND QUOTED PRICES

The reported market price at any given moment is known as the trade price. With a highly traded stock there are often several trades per second. The TAQs smallest time increment is one second. We follow Tay and Ting (2006) and discard trades outside the time interval 09:30 to 16:00 and aggregate and average trades that occur within the same second.

Quotes are also filtered the same way. For each interval, all bid and ask quotes are averaged. There is a potential problem by using quoted prices at time t together with the actual trade price at time t. A quoted price does not necessarily affect the trade price at the same instance in time. There is probably a time lag between a quoted price and the affected trade price. This is a major part of market microstructure research. Nevertheless, we do not pursue a method to match trade quotes and trade prices. Again, we let the neural network handle this data with the nonlinear relationship characteristics in mind.

4 MODELS

The following section is dedicated to model choice and the calibration of these. Both the $ARIMA$ framework and the MLP framework have several model factors that need customization, calibration and estimation before they can be utilized for time series forecasting.

4.1 CALIBRATION OF ARIMA

We will use 372 observations in our selected period for each stock to estimate a model which best describes the underlying data generating process for each stock. This will be done by cross validation. Two $ARIMA$ models for each stock will be chosen; the best model according to AIC and the best model according to BIC. Once these models have been chosen we use them on the next 180 observations. We will then choose the single best model, based on MSE, to pursue the prediction of the 60 out-of-sample observations. This will be done by a rolling one step ahead forecast. We will take the first difference of the logarithm of our four stock prices, rather than the first difference of the series, since this is more likely to be covariance stationery:

$$(32) \qquad y_t = \ln x_t - \ln x_{t-1}$$

Since our predictions will be based on y_t rather than $\ln x_t$, an $ARIMA(p, 1, q)$ for $\ln x_t$ will be the same as an $ARMA(p, q)$ for y_t.

As we can see from Table A.7 in Appendix A, there are generally low values for both $\rho_h$and ϕ_{1h}. For all four stocks there are few lags which bear statistical significance according to ACF and $PACF$ (There are some significant observations at higher lags, but these are discarded because of their position so far behind in time and their random occurrence). This makes is difficult to identify models using ACF and $PACF$ (cf. Table 1).

As a consequence, we will use the AIC and the BIC to help us with the identification process. We calculate the AIC and BIC for each stock for $ARMA(p, q)$, using $p = 0,1, \dots ,12$ and $q = 0,1, \dots ,12$. We have chosen these ranges of p and q, to best

match the procedure used for in the calibration of the MLP models. Table 2 shows the summarized results from the estimation from the first 372 observations:

TABLE 2 – RESULTS FROM ARMA ESTIMATION

STOCK		p	q	AIC	BIC
XOM	BEST AIC	3	2	-4151.37	-4123.94
	BEST BIC	1	0	-4144.76	-4133.00
PG	BEST AIC	0	0	-3954.22	-3946.38
	BEST BIC	0	0	-3954.22	-3946.38
FRO	BEST AIC	2	3	-2968.94	-2941.51
	BEST BIC	0	0	-2968.79	-2960.95
GS	BEST AIC	4	3	-4246.93	-4211.66
	BEST BIC	1	0	-4246.75	-4234.99

COMMENT: Table showing the results from the initial 360 observations in the data set. AIC and BIC values and parameters for the autoregressive and the moving average part are presented by p and q, respectively.

These seven models[8] are then used on the validation set consisting of 180 observations. Table A.1 to A.3 in Appendix A shows elaborate results[9]. The favored ARMA model for each stock, based on least MSE from the validation set, is presented in Table 3. These are thus the models we will use on the prediction set.

TABLE 3 – FINAL ARMA MODELS FROM CROSS VALIDATION

STOCK	XOM	PG	FRO	GS
MODEL	$AR(1)$	RW	$ARMA(2,3)$	$ARMA(4,3)$

COMMENT: Table showing the final $ARMA$ models which will be used in the empirical analysis on the 60 out-of-sample observations.

[8] Both AIC and BIC suggests the same model for PG, an $ARMA(0,0)$ i.e. a RW.

[9] We observe that not all coefficients are significantly different from zero. It would thus be desirable to remove the insignificant coefficients in the following selection and for the prediction set. We have, however, not pursued this approach because our main focus in this paper is on MLP models.

4.2 CALIBRATION OF NEURAL NETWORK

Building a neural network involves several steps. We assume a three layered MLP structure where all nodes are fully connected to each other because most neural networks applied in investment decisions adopt this architecture (Trippi and Lee, 1996). With the layer topology of the model set, we must select appropriate input and output values. In addition, we must find a suitable number of nodes in the hidden layer and set appropriate values for the learning rate and momentum term in the networks learning algorithm.

4.2.1 INPUT VARIABLES

In order to obtain a neural network model that predicts well, we must decide which input variables to use. We will use logarithmic stock returns as the main input:

$$(33) \qquad y_t = \ln x_t - \ln x_{t-1}$$

In addition we introduce a ratio that use the bid and ask quotes mentioned in Section 3. We define this ratio as:

$$(34) \qquad BAR_t = \frac{A_t - x_t}{A_t - B_t}$$

Where A_t is the ask quote, B_t is the bid quote and x_t is the trade price at time t. The ratio returns a value in the range $[0,1]$, and gives us an indication of how biased the spread between the bid and ask quote is. A ratio close to one indicates that the actual trade is close to the bid and vice versa. The reason for using this ratio is to feed the network with a variable that may indicate pressure to sell the stock (close to one) or buy the stock (close to zero). Now that we have the input variables, we need to settle on how many lags to use for each input variable.

Selecting too few lags could result in the loss of valuable information. Then again, too many lags could provide the neural network with noise. ACF and $PACF$ would indicate dependence between lags the same way they where used for the $ARIMA$ model. We saw, however, that both the ACF and the $PACF$ did not provide conclusive evidence of autocorrelation between different lags, possibly because of

the nonlinear properties of the time series. Sheng et al. (2003) propose that the number of input nodes of a neural network for nonlinear time series prediction can be taken as an integer just greater than or equal to the correlation dimension. Samarasinghe (2007) suggests a partial mutual information quantity as an indicator for input lags. Both methods are theoretically beyond the scope of this paper. We will use a fixed set of 12 lags for the input variables, the reason being that neural networks have a powerful ability to detect complex nonlinear relationships among a number of different variables (Kaastra and Boyd, 1996). We assume the model will manage to segregate significant lags and less significant lags by distributing weights accordingly (i.e. small weight values for less significant lags and vice versa)[10].

4.2.2 HIDDEN NODES

The greater the number of weights relative to the size of the training set, the greater the ability to memorize idiosyncrasies of individual observations. In other words, the generalization capability diminishes with increasing number of layers and weights. Neural networks with continuous nonlinear activation functions (like the logistic or hyperbolic tangent) need only one hidden layer with an arbitrarily large number of units to fit any nonlinear real function (Hornik et al., 1989). A popular method for finding the optimal number of nodes so that the model possess generalization capabilities is called *pruning* (Thimm and Fiesler, 1997); select a model with a large number of hidden units and train the model on a test set of data. Then subsequently reduce the model by iterating through the same set of data. All the models are pitted against each other with a validation data set. The model with the least error prevails and is used for prediction. This method bears similarity to the statistical method of cross validation which we utilize for our $ARMA$ calibrations. Some articles suggest other heuristic methods. Blum (1992) recommends the hidden layer size should be somewhere between the size of the input and output layer. Swingler (1996) says that one would *'never require more than twice the number of hidden nodes as you have inputs'*. Boger and Guterman (1997) specify an approach using principal component

[10] An optimal selection method for the choice of lags would be to try every conceivable permutation of all lags and test an immense number of models. This is not only extremely time consuming but also unfeasible. Let us consider a maximum of 12 lags of y_t and BAR_t. If we could construct a model for every combination, this would correspond to $(2^{12})^2 - 1$ combinations, or approximately 17 million different models!

analysis where the number of hidden nodes equals the number of principal components needed to capture around 80 percent of the variance in the input data set. The main problem with these approaches is that they do not take training iterations, error in output or the complexity of the target value (actual time series) into consideration. These are, in our view, important factors that play a pivotal role in deciding the number of hidden nodes.

We will use inverse pruning. Starting with one node, we increase the number of nodes, one at a time, until we reach the number of input nodes. We validate these models using the validation set and pick the model structure with the lowest mean square error (MSE). This model is then used to predict the time series in the same manner as our proposed $ARMA$ models. To avoid random weights that lay near a local minimum point on the error plane, we train the networks 20 times.

4.2.3 ACTIVATION FUNCTIONS AND SCALING OF DATA

Raw input data is usually scaled between the maximum and minimum values of the activation functions used in the nodes. This is to make sure that the incoming weighted sum is a reasonable size. This is also the case with values that are very small. Stock returns measured per five minutes are relatively small, and scaling these between zero and one makes a MLP model handle the input values better. A standard method of scaling is to use a linear transformation from y_t to y_t':

$$y_t' = f_{min} + (f_{max} - f_{min}) \times \frac{(y_t - y_{min})}{(y_{max} - y_{min})} \tag{35}$$

Where f_{min} and f_{max} are the minimum and maximum values of the activation function whereas y_{max} and y_{min} are the corresponding extreme values in the input set. Below are frequency histograms of historical stock returns from all four stocks. We see that the empirical distributions have relatively high kurtosis[11] (i.e. leptokurtic). Klimasauskas (1993) states that if a neural network is to learn average behavior the logistic activation function should be used. On the other hand, if

[11] Many asset pricing models, like the Black and Scholes framework for option pricing and the capital asset pricing model, assume that returns are normally distributed. Some empirical research have found evidence suggesting that the return distribution have fatter tails (higher kurtosis) than the standard normal distribution and follow a type of hyperbolic distribution (see Reimann 2005).

learning involves deviations from the average, the hyperbolic tangent function works best. The fact that the returns in our case seem to be mirrored about an axis makes the hyperbolic tangent best suited for our neural networks. The values for f_{min} and f_{max} take the value -1 and 1 *in the limit*. Because it is improbable to find returns in the limit of the tails of the distribution we to use -0.9 and 0.9 to scale input values and desired output.

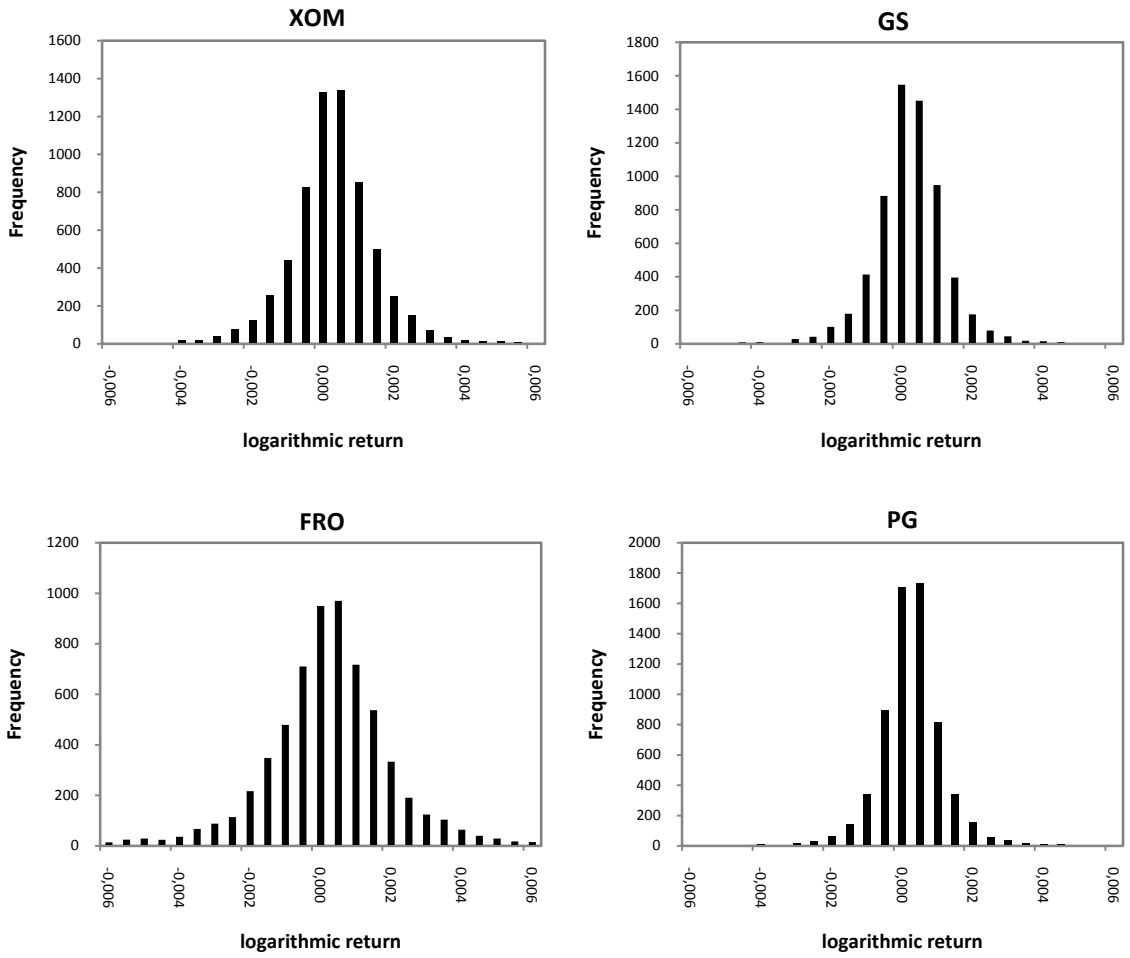


FIGURE 8 – STOCK RETURN DISTRIBUTIONS: Return distribution charts for the four stocks. The charts show that the empirical distribution has fatter tails than the standard Gaussian distribution.

4.2.4 OUTPUT NODES

The number of output nodes for time series prediction is just one for a simple MLP. This MLP structure is equivalent to (19). Lagged values of y_t and BAR_t would be transferred through the model and the outcome would be a decimal number

between -1 and 1 using the hyperbolic tangent function. This output would then be transformed back to the original time series' scale for calculation of MSE. Let us denote this model $MLP(i,j,k)_{input}^{func}$ where i, j and k reflect the number of input, hidden and output nodes. The superscript refers to the activation function in the output node. The subscript refers to the input data which is either $P = y_t'$, $B = BAR_t$ or $P, B = y_t'$ and BAR_t. Thus, a model with 24 input nodes, 12 hidden nodes, single output node and both price series and the BAR_t ratio as input is denoted $MLP(24,12,1)_{P,B}^{tanh}$.

The general MLP structure has, however, k output nodes. A tempting model structure could be a range of output nodes that correspond to an interval or a specific constant multiplied with the standard deviation of the historical return. This form of data categorization could improve the learning process, providing a range of return intervals instead of a decimal value between zero and unity. Making the output nodes binary with the choice of one or zero using a threshold function would make sense from a computer science perspective; computers work best with binary values. Consider the vector of K output nodes where the kth node contains the sum s_k *before* any activation function is used:

$$S = [s_1 \quad s_2 \quad \cdots \quad s_K] \tag{36}$$

With a binary threshold function the kth node would have the following output and corresponding vector:

$$c_k^{bin} = \begin{cases} 1 \text{ if } s_k = \max(S) \\ 0 \text{ else} \end{cases} \tag{37}$$

This would give us an output vector

$$C^{bin} = [c_1^{bin} \quad c_2^{bin} \quad \cdots \quad c_K^{bin}] \tag{38}$$

and a corresponding desired output vector given by:

$$C^{bin^*} = [c_1^{bin^*} \quad c_2^{bin^*} \quad \cdots \quad c_K^{bin^*}] \tag{39}$$

Using the binary vector C^{bin}, four intervals and the standard derivation of the sample set of logarithmic returns $\hat{\sigma}$ and the sample mean return $\hat{\mu}$, we can categorize and calculate the predicted return. Let a be a positive real number. The following four binary sequences correspond to four chosen intervals for the stock return where the subscript of C^{bin} refers to the specific interval relative to the mean:

$$(40) \qquad C_{-2}^{bin} = [1 \quad 0 \quad 0 \quad 0] \quad \Rightarrow \quad y_t < \hat{\mu} - a\hat{\sigma}$$

$$(41) \qquad C_{-1}^{bin} = [0 \quad 1 \quad 0 \quad 0] \quad \Rightarrow \quad y_t < \hat{\mu} \geq \hat{\mu} - a\hat{\sigma}$$

$$(42) \qquad C_{+1}^{bin} = [0 \quad 0 \quad 1 \quad 0] \quad \Rightarrow \quad y_t \geq \hat{\mu} < \hat{\mu} + a\hat{\sigma}$$

$$(43) \qquad C_{+2}^{bin} = [0 \quad 0 \quad 0 \quad 1] \quad \Rightarrow \quad y_t \geq \hat{\mu} + a\hat{\sigma}$$

We set $a = 0.75$ which approximately puts 40 percent of the observations in each of the intervals $[-1]$ and $[+1]$ and 10 percent in each of the intervals $[+2]$ and $[-2]$. The MSE in each output node for training pattern n is calculated using:

$$(44) \qquad e_k^n = \frac{1}{2}\left(c_k^{bin^*} - c_k^{bin}\right)^2$$

However, the intervals raise problems for the calculation of MSE when we are comparing results with the other models. To deal with this, we take the arithmetic mean for (41) and (42). For the outer intervals we use $a = 1.0$ to arrive at a fixed point not too far into the tails of the distribution of returns. Let us denote this model $MLP(i, j, 4)_{input}^{bin}$ with the same characteristics as the single-output model. The activation functions in the hidden nodes remain the same as in $MLP(i, j, 1)_{input}^{tanh}$ (i.e. the hyperbolic tangent).

4.2.5 CALIBRATION OF THE LEARNING ALGORITHM

There are, unfortunately, few systematic ways of selecting optimal values for the learning rate and momentum parameter. As a consequence, these values are usually chosen through experimentation (Zhang et al., 1998). Sharda and Patil (1992) suggest combining three different learning rates with three different momentum values; 0.1, 0.5 and 0.9. Tang et al. (1991) argue that high learning rates combined with low momentum should be used with less complex data, and vice versa.

Using a mixture of the heuristics mentioned above, we find by trial and error a learning rate of 0.085 combined with a momentum value of 0.8. Other combinations we have tried lead to oscillation and unstable learning algorithms. We therefore choose to use 0.085 for η and 0.8 for γ.

4.2.6 NUMERICAL REVIEW

We will use this section to present an example from the data set to show how the feed forward procedure and the learning algorithm works when put into practice. Consider an arbitrarily chosen model $MLP(2,2,1)_B^{tanh}$. Consider also the following input vector with two lags of the stock return y_t:

$$(45) \qquad A = [0.77 \quad 0.81]$$

Randomly selected starting weights:

$$(46) \qquad W^A = \begin{bmatrix} \phi_{11}^A & \phi_{12}^A \\ \phi_{21}^A & \phi_{22}^A \end{bmatrix} = \begin{bmatrix} 0.19 & -0.18 \\ 0.13 & 0.16 \end{bmatrix}$$

$$(47) \qquad W^B = [\phi_{11}^B \quad \phi_{12}^B] = [0.14 \quad -0.02]$$

with corresponding bias weights:

$$(48) \qquad \phi_{01}^A = [0.28] \quad , \quad \phi_{02}^A = [0.14] \quad \text{and} \quad \phi_{01}^B = [0.08]$$

Sending the weighted lagged values of y_t to each of the two hidden nodes will give us the following hidden node sums before the activation function is used:

$$(49) \qquad s_{j=1} = \phi_{01}^A + \phi_{11}^A a_1 + \phi_{21}^A a_2 \quad \text{and} \quad s_{j=2} = \phi_{02}^A + \phi_{12}^A a_1 + \phi_{22}^A a_2$$

These expressions can be thought of as linear regressions of y_t. Sending s_1 and s_2 through the nonlinear transfer function f transforms the signal to a value between -1 and 1. These results are then weighted and sent to the output node. This gives us the following expression before any activation function is used on the node sum:

$$(50) \qquad s_{k=1} = \phi_{01}^B + \phi_{11}^B f(s_{j=1}) + \phi_{21}^B f(s_{j=2})$$

This is a linear regression using the nonlinear functions as explanatory variables. The result is put through an activation function and compared to the actual value of y_t.

Looking at a small network like this, we can see how feed forward networks bear resemblance to autoregressive models and linear regressions. The transfer functions introduce a nonlinear element which makes ANNs more versatile.

Parameters in linear regressions are usually estimated by ordinary least square (OLS) problems that admit a closed-form solution. In contrast, nonlinear least squares problems like estimating the parameters of an ANN have to be solved by an iterative procedure. ANN weights are usually initiated as random values between -0.3 and 0.3 and subsequently adjusted by an algorithm. Using the data above we will now present a numerical example of how our weights are adjusted using the back-propagation algorithm.

Consider the output after sending the values from vector A through the network:

$$(51) \qquad c_1 = f\left(\phi_{01}^B + \phi_{11}^B f\left(s_{j=1}\right) + \phi_{21}^B f\left(s_{j=2}\right)\right)$$

Substituting for the weights and hidden node sums with values from above:

$$(52) \qquad c_1 = f(0.1112) = \frac{e^{2\times 0.1112} - 1}{e^{2\times 0.1112} + 1} = 0.1107$$

The actual value of y_t is -0.0023. This yields an error of:

$$(53) \qquad e_1 = \frac{1}{2}(-0.0023 - 0.1107)^2 = 0.0064$$

Using (24) and (25) together with our chosen values for η and γ we get the following gradient for the output layer:

$$(54) \qquad \delta_{k=1} = (-0.0023 - 0.1107) \times (1 - f(0.1112)^2) = -0.112$$

$$(55) \qquad \Delta_1\, \phi_{11}^B = 0.085 \times -0.112 \times 0.1112 + 0.8 \times 0 = -0.001$$

The change in weight ϕ_{11}^A depends on the gradient calculated above. Using (25) we calculate the change:

$$(56) \qquad \delta_{j=1} = (-0.112 \times 0.14) \times (1 - f(0.2805)^2) = -0.015$$

$$(57) \qquad \Delta_1\, \phi_{11}^A = 0.085 \times -0.015 \times 0.2805 + 0.8 \times 0 = -0.0004$$

The weight changes for ϕ_{11}^{B} and ϕ_{21}^{A} are shown in Figure 10 below.

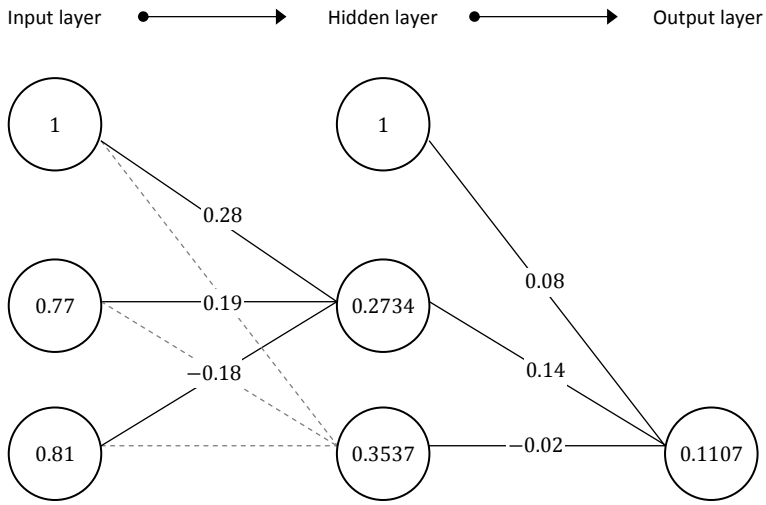


FIGURE 9 – FEED FORWARDING INPUT VALUES: Shows how the data flows in the network. Some weights are in dotted lines to make it easier to see the flow. The node values are after activation of nonlinear function.

The weight changes for ϕ_{11}^{B} and ϕ_{21}^{A} are shown in Figure 10:

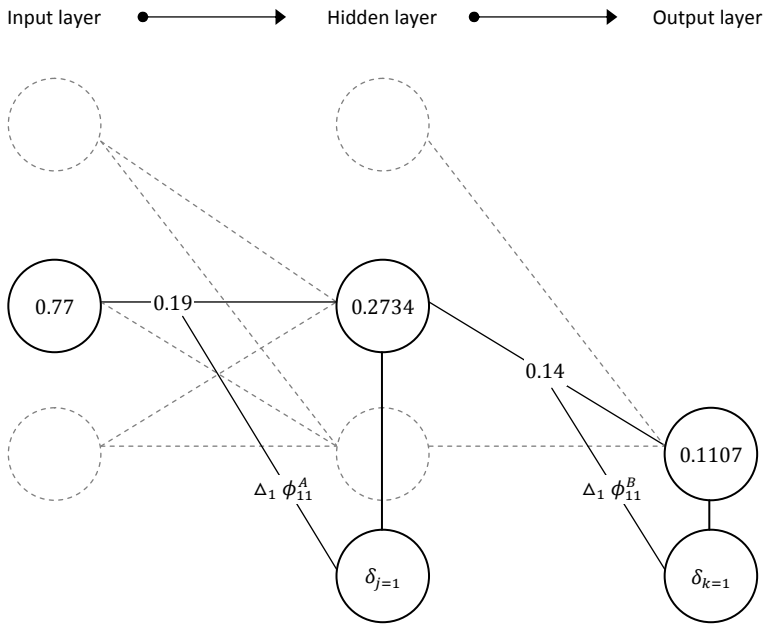


FIGURE 10 – BACK-PROPAGATION: A representation of the weight changes using the back-propagation algorithm.

This example shows how input data flows through the system and how the estimation procedure of the models parameters works. In the next section we will use this model on the whole data set and try to predict stock returns in five minute intervals.

5 EMPIRICAL ANALYSIS

In this chapter we will discuss the results, and discuss our ANNs as suitable models for prediction of our four selected time series. To determine this in a satisfyingly manner we cannot just look at the results from our ANNs. We will therefore compare the selected ANNs to the RW and the $ARMA$ models.

5.1 NEURAL NETWORKS

We will first summarize the results from the empirical analysis based on our feed forward neural networks, and compare them to a RW. We only present the best ANNs in this section with reference to more detailed tables in Appendix A. There are three different versions of the model depending on input variables; only price series, only the BAR series and both series.

5.1.1 NEURAL NETWORK A – ONE STANDARD OUTPUT NODE

The results from the $MLP(i,j,1)^{tanh}$ model used on the four stocks are summarized in Table 4. For XOM and PG, the best ANN used both the price series and our BAR ratio. For these two stocks, the ANNs did not beat the *RW* based on the *MSE* measure, the *U*-statistic or the S-statistic.

TABLE 4 – NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS

STOCK	XOM	PG	FRO	GS
MODEL	$MLP(24,18,1)^{tanh}_{P,B}$	$MLP(24,8,1)^{tanh}_{P,B}$	$MLP(12,8,1)^{tanh}_{B}$	$MLP(12,4,1)^{tanh}_{P}$
MSE	1.56E-06	6.41E-07	2.22E-06	4.76E-07
MSE RW	8.03E-07	4.38E-07	6.94E-06	7.64E-07
U-STAT	1.62	4.70	3.53	1.66
S-STAT STOCK	0.40	0.30	0.63	0.53
S-STAT RW	0.60	0.60	0.60	0.60

COMMENT: Detailed tables for all four stocks are found in Appendix A from Table A.1 to A.4.

For FRO and GS, the ANN used solely the BAR ratio and the price series, respectively. As we can see, the ANNs beat the RW based on MSE. However, looking at the U-statistic, we can not infer that these ANNs satisfyingly predict the stock returns from a statistically perspective. In addition, the S-statistics are worse or marginally better than that of RW. We can therefore not conclude that these ANNs are better than a RW.

5.1.2 NEURAL NETWORK B – FOUR BINARY OUTPUT NODES

As for the single output node model, there are three main models depending on the time series input variables. Again, we show the results in Table 5 using the model structure $MLP(i,j,4)^{bin}$. For XOM, PG and GS the ANNs do not beat RW. The U-statistics are larger than one and the S-statistics are lower or the same as that of RW. We also notice that the binary ANNs use the price series as their only input, discarding our BAR ratio.

TABLE 5 – NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS

STOCK	XOM	PG	FRO	GS
MODEL	$MLP(12,4,4)_P^{bin}$	$MLP(12,3,4)_P^{bin}$	$MLP(12,10,4)_P^{bin}$	$MLP(12,6,4)_P^{bin}$
MSE	1.10E-06	1.02E-06	6.85E-06	5.22E-06
MSE RW	8.03E-07	4.38E-07	6.94E-06	7.64E-07
U-STAT	1.32	1.28	1.17	2.78
S-STAT STOCK	0.60	0.60	0.52	0.62
S-STAT RW	0.60	0.60	0.60	0.60

COMMENT: Detailed tables for all four stocks are found in Appendix A from Table A.5 to A.8.

FRO is the only stock where our binary ANN beat the RW. The U-statistic is close to unity, but the S-statistic is close to 50 percent (lower than RW). This means that one might as well toss a coin to predict the direction. Like above, we cannot infer that our ANNs perform better than a RW.

5.2 COMPARATIVE ANALYSIS

Since we could not make a certain conclusion about how the ANNs preformed in comparison to a RW, we will in this section compare the results from using the $ARMA$ models selected in Section 4.1 against the ANNs. We use the best $ARMA$ models on the last 60 observations in our data sample. The results are summarized in Table 6 below.

TABLE 6 – ARIMA AND NETWORK A & B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS

STOCK	XOM	PG	FRO	GS
MODEL	$MLP(24,18,1)^{tanh}_{P,B}$	$MLP(24,8,1)^{tanh}_{P,B}$	$MLP(12,8,1)^{tanh}_{B}$	$MLP(12,4,1)^{tanh}_{P}$
MSE	1,56E-06	6,41E-07	2,22E-06	4,76E-07
S-STAT	0,40	0,30	0,63	0,53
MODEL	$MLP(12,4,4)^{bin}_{P}$	$MLP(12,3,4)^{bin}_{P}$	$MLP(12,10,4)^{bin}_{P}$	$MLP(12,6,4)^{bin}_{P}$
MSE	1.10E-06	1.02E-06	6.85E-06	5.22E-06
S-STAT	0.60	0.60	0.52	0.62
MODEL	$AR(1)$	RW	$ARMA(2,3)$	$AR(1)$
MSE	7,97E-07	4,38E-07	6,64E-07	7,15E-07
S-STAT	0,60	0,60	0,330	0,62

COMMENT: Results are based on the last 60 observations from the data set.

For our single output models we observe, with the exception of FROs S-statistic and GSs MSE, that the $ARMA$ models are better than our ANNs. Comparing the binary models to the $ARMA$ models we can see, with the exception of FROs S-statistics, that The $ARMA$ models are equal or better than our ANNs.

6 CONCLUSION

We started this paper with a proposal that nonlinear frameworks might provide better predictions than linear frameworks because of the nonlinear properties of stock price time series. We conclude, however, that the models proposed by us cannot predict the stock price return in a satisfyingly matter compared to a RW and selected $ARMA$ models. On the other hand, we see strong potential in these models. They are very flexible and we have only touched their most basic structure and learning algorithm.

Several papers on neural networks have concluded that their models predict better than standard linear models (see Adya and Collopy (1998) for a comprehensive study). It would be both unfair and wrong to try to compare our results with theirs. Our models are trivial compared to some of the models applied in those papers. We have, however, tried to introduce the building blocks of a framework that we see as a big part in the future of time series analysis in an easier and detailed fashion. We have tried to carefully explain the steps in making a simple feed forward neural network and in the same process, gotten rid of some of the mysteries surrounding these types of models.

For future research we would recommend researchers to expand on our simple models using different input data and more complex network structures. Using traded volume or the time series of a stock index used as a market proxy could improve these models. Changing the architecture of the network might yield a better network. A *recurrent neural network* is an example a more exotic architecture where one or more of the output values are sent back into hidden layers. Another interesting approach is a hybrid model of $ARIMA$ and ANNs proposed by Zhang (2003). The time series is first used in the $ARIMA$ framework and the residual (assumed to be nonlinear) is used in the ANN framework.

A TABLES AND FIGURES

TABLE A.1 – ESTIMATED ARMA MODEL FOR EXXON MOBILE

XOM						
ARMA(3,2)	AR1	AR2	AR3	MA1	MA2	CONST.
Coefficients	1.7973	-1.2473	0.2189	-1.6241	0.9612	-1.00E-04
S.e.	0.0709	0.0896	0.0569	0.0561	0.0395	1.00E-04
Coef/S.e.	25.3	-13.9	3.8	-29.0	24.3	-1.0
$Sigma^2$	8.01E-07					
Log likelihood	2082.68					
AR(1)	AR1	CONST.				
Coefficients	0.2121	-1.00E-04				
s.e.	0.0508	1.00E-04				
Coef/s.e.	4.2	-1.0				
$Sigma^2$	8.35E-07					
Log likelihood	2075.38					

COMMENT: Result summary from R.

TABLE A.2 – ESTIMATED ARMA MODEL FOR FRONTLINE

FRO						
ARMA(3,2)	AR1	AR2	MA1	MA2	MA3	CONST.
Coefficients	1.0613	-0.7041	-1.0209	0.5900	0.1465	0.00E+00
S.e.	0.1934	0.1613	0.1961	0.1890	0.0585	3.00E-04
Coef/S.e.	5.4876	-4.3652	-5.2060	3.1217	2.5043	0
$Sigma^2$	1.93E-05					
Log likelihood	1491.47					

COMMENT: Result summary from R.

TABLE A.3 – ESTIMATED ARMA MODEL FOR GOLDMAN SACHS

XOM

ARMA(3,2)	AR1	AR2	AR3	AR4	MA1	MA2	MA3	CONST.
Coefficients	0.1015	-0.0419	0.9262	-0.1856	0.0447	0.0314	-0.9621	0.00E+00
S.e.	0.0606	0.0393	0.0379	0.0522	0.0356	0.0332	0.0320	1.00E-04
Coef/S.e.	1.6749	-1.0662	24.4380	-3.5556	1.2556	0.9458	-30.0656	0
$Sigma^2$	6.07E-07							
Log likelihood	2132.47							

AR(1)	AR1	CONST.
Coefficients	0.1371	0.00E+00
S.e.	0.0516	1.00E-04
Coef/S.e.	2.6570	0
$Sigma^2$	6.35E-07	
Log likelihood	2126.37	

COMMENT: Result summary from R.

TABLE A.4 – ARMA RESULTS FROM CROSS VALIDATION

STOCK			
XOM	ARMA(3,2)	AR(1)	RW
MSE	4.56E-07	4.09E-07	6.32E-07
PG	RW		
MSE	6.64E-06		
FRO	ARMA(2,3)	RW	
MSE	6.94E-06	9.69E-07	
GS	ARMA(4,3)	AR(1)	RW
MSE	9.50E-07	1.11E-06	1.01E-06

COMMENT: Results from cross validation using selected models on the basis of AIC and BIC.

TABLE A.5 – AUTOCORRELATION COEFFICIENTS AND PARIAL AUTOCORRELATION COEFFICIENTS

	XOM		PG		FRO		GS	
h	ρ_h	ϕ_{1h}	ρ_h	ϕ_{1h}	ρ_h	ϕ_{1h}	ρ_h	ϕ_{1h}
1	0,170	0,170	0,017	0,017	0,001	0,001	0,094	0,094
	(4,21)	(4,21)	(0,43)	(0,43)	(0,03)	(0,03)	(2,32)	(2,32)
2	0,076	0,048	0,024	0,024	-0,053	-0,053	0,076	0,068
	(1,82)	(1,19)	(0,59)	(0,59)	-(1,32)	-(1,32)	(1,87)	(1,69)
3	0,091	0,072	-0,027	-0,028	0,047	0,047	0,041	0,028
	(2,17)	(1,79)	-(0,67)	-(0,69)	(1,15)	(1,16)	(0,99)	(0,69)
4	0,061	0,032	-0,040	-0,040	0,050	0,047	-0,025	-0,036
	(1,44)	(0,80)	-(0,99)	-(0,98)	(1,23)	(1,17)	-(0,60)	-(0,90)
5	0,022	-0,001	0,010	0,013	0,042	0,048	0,001	0,002
	(0,52)	-(0,04)	(0,25)	(0,32)	(1,04)	(1,18)	(0,03)	(0,04)
6	0,071	0,059	-0,104	-0,104	0,056	0,060	0,045	0,049
	(1,68)	(1,46)	-(2,57)	-(2,57)	(1,38)	(1,47)	(1,10)	(1,21)
7	0,054	0,027	-0,054	-0,054	-0,038	-0,039	-0,014	-0,020
	(1,26)	(0,67)	-(1,31)	-(1,33)	-(0,94)	-(0,95)	-(0,34)	-(0,50)
8	-0,014	-0,037	0,008	0,014	-0,076	-0,078	0,012	0,007
	-(0,32)	-(0,90)	(0,21)	(0,35)	-(1,85)	-(1,92)	(0,30)	(0,19)
9	-0,041	-0,048	-0,025	-0,028	0,001	-0,013	0,035	0,033
	-(0,95)	-(1,20)	-(0,60)	-(0,70)	(0,04)	-(0,33)	(0,86)	(0,83)
10	-0,131	-0,132	-0,043	-0,056	-0,010	-0,023	-0,113	-0,119
	-(3,07)	-(3,26)	-(1,05)	-(1,38)	-(0,24)	-(0,56)	-(2,74)	-(2,93)
11	-0,017	0,027	-0,002	0,000	0,007	0,013	-0,029	-0,016
	-(0,40)	(0,68)	-(0,04)	(0,00)	(0,18)	(0,33)	-(0,69)	-(0,39)
12	-0,011	0,004	0,036	0,029	-0,005	0,003	0,018	0,037
	-(0,27)	(0,10)	(0,88)	(0,71)	-(0,12)	(0,06)	(0,44)	(0,91)
13	0,011	0,031	0,063	0,047	-0,018	-0,004	-0,026	-0,017
	(0,25)	(0,77)	(1,54)	(1,15)	-(0,44)	-(0,09)	-(0,63)	-(0,43)
14	-0,024	-0,019	0,056	0,050	-0,071	-0,064	-0,040	-0,049
	-(0,56)	-(0,47)	(1,35)	(1,25)	-(1,71)	-(1,58)	-(0,95)	-(1,20)
15	-0,044	-0,034	0,027	0,023	0,033	0,027	0,007	0,013
	-(1,02)	-(0,85)	(0,64)	(0,57)	(0,80)	(0,68)	(0,18)	(0,32)
16	-0,068	-0,040	0,017	0,008	-0,014	-0,025	-0,015	0,005
	-(1,56)	-(0,99)	(0,41)	(0,20)	-(0,34)	-(0,63)	-(0,35)	(0,12)
17	-0,042	-0,012	-0,024	-0,023	-0,028	-0,021	0,031	0,029
	-(0,96)	-(0,30)	-(0,57)	-(0,57)	-(0,68)	-(0,51)	(0,74)	(0,73)
18	0,021	0,038	0,019	0,031	-0,009	-0,008	0,002	-0,007
	(0,49)	(0,93)	(0,46)	(0,78)	-(0,23)	-(0,19)	(0,05)	-(0,18)
19	-0,020	-0,028	-0,015	-0,001	-0,006	-0,001	0,018	0,028
	-(0,46)	-(0,70)	-(0,35)	-(0,01)	-(0,14)	-(0,03)	(0,43)	(0,70)
20	-0,051	-0,060	-0,009	0,002	0,019	0,026	-0,024	-0,041
	-(1,17)	-(1,48)	-(0,21)	(0,05)	(0,45)	(0,65)	-(0,58)	-(1,02)

COMMENT: Different lags of the autocorrelation coefficients and partial autocorrelation coefficients for the first 372 observations. Values in parentheses are t-values with a 95 % confidence interval.

TABLE A.6 – NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (XOM)

	$MLP(12,9,1)_{P}^{tanh}$	$MLP(12,9,1)_{B}^{tanh}$	$MLP(24,18,1)_{P,B}^{tanh}$	RW
MSE	2.56E-06	2.45E-06	1.56E-06	8.03E-07
U-STATISTIC	1.43	1.38	1.62	-
S-STATISTIC	0.40	0.40	0.40	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.7 – NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (PG)

	$MLP(12,10,1)_{P}^{tanh}$	$MLP(12,5,1)_{B}^{tanh}$	$MLP(24,16,1)_{P,B}^{tanh}$	RW
MSE	6.77E-07	6.55E-07	6.41E-07	4.38E-07
U-STATISTIC	4.41	4.34	4.70	-
S-STATISTIC	0.43	0.40	0.30	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.8 – NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (FRO)

	$MLP(12,10,1)_{P}^{tanh}$	$MLP(12,8,1)_{B}^{tanh}$	$MLP(24,10,1)_{P,B}^{tanh}$	RW
MSE	2.30E-06	2.22E-06	3.75E-06	6.94E-06
U-STATISTIC	2.19	3.53	5.95	-
S-STATISTIC	0.53	0.63	0.50	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.9 – NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (GS)

	$MLP(12,4,1)_P^{tanh}$	$MLP(12,5,1)_B^{tanh}$	$MLP(24,8,1)_{P,B}^{tanh}$	RW
MSE	4.76E-07	4.78E-07	8.6E-07	7.64E-07
U-STATISTIC	1.66	1.02	1.90	-
S-STATISTIC	0.53	0.57	0.47	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.10 – NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (XOM)

	$MLP(12,3,4)_P^{bin}$	$MLP(12,4,4)_B^{bin}$	$MLP(24,11,4)_{P,B}^{bin}$	RW
MSE	1.02E-06	1.10E-06	2.57E-06	4.38E-07
U-STATISTIC	1.28	1.28	1.80	-
S-STATISTIC	0.60	0.60	0.53	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.11 – NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (PG)

	$MLP(12,3,4)_P^{bin}$	$MLP(12,4,4)_B^{bin}$	$MLP(24,11,4)_{P,B}^{bin}$	RW
MSE	1.02E-06	1.10E-06	2.57E-06	4.38E-07
U-STATISTIC	1.28	1.28	1.80	-
S-STATISTIC	0.60	0.60	0.53	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.12 – NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (FRO)

	$MLP(12,10,4)_P^{bin}$	$MLP(12,10,4)_B^{bin}$	$MLP(24,20,4)_{P,B}^{bin}$	RW
MSE	6.85E-06	6.92E-06	6.97E-06	3.25E-06
U-STATISTIC	1.17	1.17	1.75	-
S-STATISTIC	0.52	0.52	0.52	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.13 – NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (GS)

	$MLP(12,6,4)_P^{bin}$	$MLP(12,7,4)_B^{bin}$	$MLP(24,10,4)_{P,B}^{bin}$	RW
MSE	5.22E-06	5.48E-06	5.78E-06	7.64E-07
U-STATISTIC	2.78	2.88	2.88	-
S-STATISTIC	0.62	0.55	0.57	0.60

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations

TABLE A.14 – DETAILED NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (XOM)

j	$MLP(12, j, 1)_{P}^{tanh}$	$MLP(12, j, 1)_{B}^{tanh}$	$MLP(24, j, 1)_{P,B}^{tanh}$	RW
1	2,66E-06	2,48E-06	0.00E-00**	8.03E-07
2	2,62E-06	2,46E-06	0.00E-00**	-
3	2,62E-06	2,47E-06	0.00E-00**	-
4	2,58E-06	2,47E-06	0.00E-00**	-
5	2,56E-06	2,46E-06	0.00E-00**	-
6	2,57E-06	2,46E-06	0.00E-00**	-
7	2,55E-06	2,48E-06	0.00E-00**	-
8	2,55E-06	2,46E-06	2,28E-06	-
9	2,56E-06*	2,45E-06*	1,91E-06	-
10	2,59E-06	2,47E-06	1,70E-06	-
11	2,57E-06	2,47E-06	1,78E-06	-
12	2,60E-06	2,46E-06	1,82E-06	-
13	-	-	3,09E-06	-
14	-	-	2,32E-06	-
15	-	-	3,10E-06	-
16	-	-	2,52E-06	-
17	-	-	2,70E-06	-
18	-	-	1,56E-06*	-
19	-	-	2,01E-06	-
20	-	-	2,68E-06	-
21	-	-	2,66E-06	-
22	-	-	2,64E-06	-
23	-	-	1,79E-06	-
24	-	-	2,72E-06	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.15 – DETAILED NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (PG)

j	$MLP(12,j,1)_{P}^{tanh}$	$MLP(12,j,1)_{B}^{tanh}$	$MLP(24,j,1)_{P,B}^{tanh}$	RW
1	7,00E-07	7,18E-07	0.00E-00**	4.38E-07
2	6,71E-07	7,15E-07	0.00E-00**	-
3	6,82E-07	6,72E-07	0.00E-00**	-
4	6,84E-07	6,56E-07	0.00E-00**	-
5	6,83E-07	6,55E-07*	8,35E-07	-
6	6,92E-07	6,70E-07	7,61E-07	-
7	6,91E-07	6,69E-07	7,64E-07	-
8	6,85E-07	6,63E-07	6,41E-07	-
9	6,77E-07	6,70E-07	7,52E-07	-
10	6,77E-07*	6,74E-07	7,55E-07	-
11	6,84E-07	6,66E-07	8,30E-07	-
12	6,83E-07	6,69E-07	7,53E-07	-
13	-	-	8,49E-07	-
14	-	-	8,39E-07	-
15	-	-	8,21E-07	-
16	-	-	6,87E-07*	-
17	-	-	9,13E-07	-
18	-	-	7,55E-07	-
19	-	-	9,68E-07	-
20	-	-	7,29E-07	-
21	-	-	7,83E-07	-
22	-	-	9,11E-07	-
23	-	-	8,69E-07	-
24	-	-	9,12E-07	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.16 – DETAILED NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (FRO)

j	$MLP(12, j, 1)_P^{tanh}$	$MLP(12, j, 1)_B^{tanh}$	$MLP(24, j, 1)_{P,B}^{tanh}$	RW
1	0.00E-00**	0.00E-00**	0.00E-00**	3.25E-06
2	2,41E-06	0.00E-00**	0.00E-00**	-
3	2,47E-06	3,09E-06	0.00E-00**	-
4	2,33E-06	3,12E-06	0.00E-00**	-
5	3,25E-06	2,54E-06	3,17E-06	-
6	2,83E-06	2,45E-06	4,13E-06	-
7	2,38E-06	2,45E-06	7,11E-06	-
8	2,76E-06	2,22E-06*	1,06E-05	-
9	2,38E-06	2,26E-06	9,27E-06	-
10	2,30E-06*	2,32E-06	3,75E-06*	-
11	2,35E-06	2,32E-06	9,95E-06	-
12	2,35E-06	2,36E-06	7,05E-06	-
13	-	-	8,75E-06	-
14	-	-	9,39E-06	-
15	-	-	7,22E-06	-
16	-	-	8,67E-06	-
17	-	-	8,73E-06	-
18	-	-	6,80E-06	-
19	-	-	7,50E-06	-
20	-	-	7,35E-06	-
21	-	-	7,18E-06	-
22	-	-	7,65E-06	-
23	-	-	8,52E-06	-
24	-	-	7,06E-06	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.17 – DETAILED NETWORK A RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (GS)

j	$MLP(12, j, 1)_{P}^{tanh}$	$MLP(12, j, 1)_{B}^{tanh}$	$MLP(24, j, 1)_{P,B}^{tanh}$	RW
1	8,00E-07	6,15E-07	0.00E-00**	7.64E-07
2	4,94E-07	1,28E-06	0.00E-00**	-
3	1,02E-06	6,13E-07	0.00E-00**	-
4	4,76E-07*	5,44E-07	0.00E-00**	-
5	5,46E-07	4,78E-07*	0.00E-00**	-
6	5,03E-07	1,01E-06	0.00E-00**	-
7	1,13E-06	9,11E-07	7,17E-07	-
8	1,01E-06	8,50E-07	6,18E-07*	-
9	9,75E-07	8,89E-07	9,34E-07	-
10	9,51E-07	8,35E-07	8,60E-07	-
11	1,03E-06	8,84E-07	9,01E-07	-
12	9,62E-07	8,83E-07	1,10E-06	-
13	-	-	9,05E-07	-
14	-	-	8,85E-07	-
15	-	-	9,89E-07	-
16	-	-	1,03E-06	-
17	-	-	1,04E-06	-
18	-	-	1,00E-06	-
19	-	-	9,43E-07	-
20	-	-	8,95E-07	-
21	-	-	9,03E-07	-
22	-	-	8,83E-07	-
23	-	-	8,99E-07	-
24	-	-	8,98E-07	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.18 – DETAILED NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (XOM)

j	$MLP(12,j,1)_{P}^{bin}$	$MLP(12,j,1)_{B}^{bin}$	$MLP(24,j,1)_{P,B}^{bin}$	RW
1	2,87E-06	2,87E-06	0.00E-00**	8.03E-07
2	3,36E-06	5,08E-06	0.00E-00**	-
3	2,91E-06	2,83E-06	0.00E-00**	-
4	1,10E-06*	2,97E-06	0.00E-00**	-
5	2,93E-06	1,18E-06*	4,07E-06	-
6	4,21E-06	3,31E-06	4,34E-06	-
7	4,26E-06	4,19E-06	4,33E-06	-
8	4,26E-06	4,36E-06	4,09E-06	-
9	4,41E-06	4,69E-06	4,52E-06	-
10	4,30E-06	4,43E-06	4,42E-06	-
11	4,31E-06	4,22E-06	4,03E-06*	-
12	4,38E-06	4,57E-06	4,46E-06	-
13	-	-	5,52E-06	-
14	-	-	4,64E-06	-
15	-	-	4,31E-06	-
16	-	-	4,45E-06	-
17	-	-	4,40E-06	-
18	-	-	4,42E-06	-
19	-	-	4,33E-06	-
20	-	-	4,57E-06	-
21	-	-	4,08E-06	-
22	-	-	4,64E-06	-
23	-	-	4,49E-06	-
24	-	-	4,63E-06	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.19 – DETAILED NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (PG)

j	$MLP(12, j, 1)_{P}^{bin}$	$MLP(12, j, 1)_{B}^{bin}$	$MLP(24, j, 1)_{P,B}^{bin}$	RW
1	2,87E-06	2,87E-06	0.00E-00**	4.38E-07
2	2,87E-06	2,87E-06	0.00E-00**	-
3	1,02E-06*	1,22E-06	0.00E-00**	-
4	2,89E-06	1,10E-06*	0.00E-00**	-
5	4,09E-06	1,11E-06	0.00E-00**	-
6	4,32E-06	4,45E-06	0.00E-00**	-
7	4,67E-06	4,30E-06	0.00E-00**	-
8	4,70E-06	4,29E-06	2,69E-06	-
9	4,25E-06	4,42E-06	2,65E-06	-
10	4,31E-06	4,30E-06	5,53E-06	-
11	4,26E-06	4,44E-06	2,57E-06*	-
12	4,67E-06	4,43E-06	6,87E-06	-
13	-	-	5,77E-06	-
14	-	-	6,20E-06	-
15	-	-	5,73E-06	-
16	-	-	6,08E-06	-
17	-	-	6,83E-06	-
18	-	-	4,59E-06	-
19	-	-	6,67E-06	-
20	-	-	7,08E-06	-
21	-	-	5,58E-06	-
22	-	-	6,73E-06	-
23	-	-	6,41E-06	-
24	-	-	7,73E-06	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.20 – DETAILED NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (FRO)

j	$MLP(12, j, 1)_{P}^{bin}$	$MLP(12, j, 1)_{B}^{bin}$	$MLP(24, j, 1)_{P,B}^{bin}$	RW
1	6.96E-06	6.93E-06	0.00E-00**	3.25E-06
2	6.93E-06	7.02E-06	0.00E-00**	-
3	6.99E-06	7.01E-06	0.00E-00**	-
4	6.92E-06	7.02E-06	0.00E-00**	-
5	6.88E-06	7.01E-06	0.00E-00**	-
6	6.86E-06	6.94E-06	0.00E-00**	-
7	6.88E-06	6.93E-06	0.00E-00**	-
8	6.88E-06	6.93E-06	0.00E-00**	-
9	6.88E-06	6.95E-06	7.23E-06	-
10	6.85E-06*	6.92E-06*	7.12E-06	-
11	6.88E-06	6.95E-06	7.41E-06	-
12	7.10E-06	6.94E-06	7.35E-06	-
13	-	-	7.28E-06	-
14	-	-	7.31E-06	-
15	-	-	7.35E-06	-
16	-	-	7.46E-06	-
17	-	-	7.41E-06	-
18	-	-	7.23E-06	-
19	-	-	7.35E-06	-
20	-	-	6.97E-06*	-
21	-	-	7.44E-06	-
22	-	-	7.22E-06	-
23	-	-	0.00E-00**	-
24	-	-	0.00E-00**	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

TABLE A.21 – DETAILED NETWORK B RESULTS ON OUT-OF-SAMPLE OBSERVATIONS (GS)

j	$MLP(12,j,1)_{P}^{bin}$	$MLP(12,j,1)_{B}^{bin}$	$MLP(24,j,1)_{P,B}^{bin}$	RW
1	6.31E-06	6.31E-06	0.00E-00**	7.64E-07
2	5.62E-06	5.87E-06	0.00E-00**	-
3	6.31E-06	5.85E-06	0.00E-00**	-
4	5.91E-06	5.85E-06	0.00E-00**	-
5	5.79E-06	6.10E-06	0.00E-00**	-
6	5.22E-06*	5.73E-06	0.00E-00**	-
7	5.65E-06	5.48E-06*	0.00E-00**	-
8	5.82E-06	5.82E-06	5.82E-06	-
9	5.88E-06	6.10E-06	6.10E-06	-
10	5.82E-06	6.10E-06	5.78E-06*	-
11	5.82E-06	5.82E-06	5.82E-06	-
12	5.82E-06	5.83E-06	6.10E-06	-
13	-	-	0.00E-00**	-
14	-	-	0.00E-00**	-
15	-	-	0.00E-00**	-
16	-	-	0.00E-00**	-
17	-	-	0.00E-00**	-
18	-	-	0.00E-00**	-
19	-	-	0.00E-00**	-
20	-	-	0.00E-00**	-
21	-	-	0.00E-00**	-
22	-	-	0.00E-00**	-
23	-	-	0.00E-00**	-
24	-	-	0.00E-00**	-

COMMENT: Results are based on 60 observations (10 %) from the data set. The analysis was conducted using 20 initializations of random weights. The total number of epochs was 2000.

* Best network in training batch

** Weights did not converge and network is unstable – no final solution

B DERIVATION OF BACK-PROPAGATION ALGORITHM

For a complete theoretical review and numerical examples, see Rumelhart and McClelland (1986) or Samarasinghe (2007). For the theoretical derivation of the momentum term γ, see for example Hagiwara (1992). Let c_k be the output from output node k and c_k^* be the desired output. The output node error in training pattern n is half of the squares of the difference between c_k^* and c_k:

$$(B.1) \qquad e_k^n = \frac{1}{2}(c_k^* - c_k)^2$$

With K output nodes and N training patterns, the sum of squared errors is:

$$(B.2) \qquad E^n = \sum_{k=1}^{K} e_k^n \quad \text{where} \quad n = 1,2,\dots,N$$

This gives us an average squared error:

$$(B.3) \qquad E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} e_k^n$$

For the sake of simplicity, let us for the rest of the proof relax the notation and drop the superscript n which indicate training pattern. The change in the weight from node j to node k should be proportional to the negative of the slope of the error with respect to the weight where η is the learning rate:

$$(B.4) \qquad \phi_{jk}^B = \phi_{jk}^B - \eta\frac{\partial e_k}{\partial \phi_{jk}^B}$$

Applying the chain rule, we can express the partial derivative in $(B.4)$ as a product of partial derivatives:

$$(B.5) \qquad \frac{\partial e_k}{\partial \phi_{jk}^B} = \frac{\partial e_k}{\partial s_k}\frac{\partial s_k}{\partial \phi_{jk}^B}$$

Where s_k is the input sum from node k before activation. Rearranging simplifies the second derivative term in $(B.5)$ to the activated sum b_j for node j:

$$(B.6) \qquad \frac{\partial s_k}{\partial \phi_{jk}^B} = \frac{\partial}{\partial \phi_{jk}^B} \sum_{k=1}^{K} \phi_{jk}^B \, c_k = s_k$$

We define δ_k as:

$$(B.7) \qquad \delta_k = -\frac{\partial e_k}{\partial s_k}$$

Now we have an expression for the weight change in terms of the delta for the destination unit. Substituting $(B.6)$ and $(B.7)$ into $(B.4)$ yields:

$$(B.8) \qquad \phi_{jk}^B = \phi_{jk}^B + \eta \delta_k s_k$$

Using the chain rule again we can express delta as a product of partial derivatives:

$$(B.9) \qquad \delta_k = -\frac{\partial e_k}{\partial s_k} = -\frac{\partial e_k}{\partial b_j} \frac{\partial b_j}{\partial s_k}$$

The second of the derivatives in $(B.9)$ is just the derivative of the activation function:

$$(B.10) \qquad \frac{\partial b_j}{\partial s_k} = f'(s_k)$$

For the first derivative term in $(B.9)$ there are two cases; k is an output node or k is a hidden node. If the first case is true, the derivative is just the difference of the output and desired output.

$$(B.11) \qquad \frac{\partial e_k}{\partial a_j} = -(c_k^* - c_k)$$

It follows by substitution:

$$(B.12) \qquad \delta_k = (c_k^* - c_k) f'(s_k)$$

The general weight change for a weight between the output layer and a hidden layer is then:

$$(B.13) \qquad \phi_{jk}^B = \phi_{jk}^B + \eta (c_k^* - c_k) f'(s_k) s_k$$

If the weight is between the output layer and the hidden layer we must account for the error from the layer above. We simply use the chain rule once more to express

the first derivative in $(B.9)$ as a sum of products of partial derivatives. Using $(B.7)$, the expression simplifies to the negative of the sum of the products of the deltas of the units in the layer above and the weights connecting to those units:

$$(B.14) \qquad \sum_{k=1}^{K} \frac{\partial e_k}{\partial s_k} \frac{\partial s_k}{\partial b_j} = \sum_{k=1}^{K} \frac{\partial e_k}{\partial s_k} \phi_{jk}^{B} = -\sum_{k=1}^{K} \delta_k \phi_{jk}^{B}$$

Substituting yields:

$$(B.15) \qquad \delta_j = \left(\sum_{k=1}^{K} \delta_k \phi_{jk}^{B} \right) f'(s_j)$$

The weight change for weights between the input layer and the hidden layer is consequently:

$$(B.16) \qquad \phi_{ij}^{A} = \phi_{ij}^{A} + \eta \left(\sum_{k=1}^{K} \delta_k \phi_{jk}^{B} \right) f'(s_j) s_j$$

The momentum term is added to equation $(B.8)$ and $(B.16)$ and is defined as a factor γ times the weight change in the last training iteration. Thus, the final expression for the weight change between the hidden layer and the output layer is:

$$(B.17) \qquad \Delta_t \, \phi_{jk}^{B} = \eta \delta_k s_k + \gamma \, \Delta_{t-1} \, \phi_{jk}^{B} \quad \text{where} \quad \delta_k = (c_k^* - c_k) f'(s_k)$$

Equally, the weight change between the hidden layer and the input layer is given by:

$$(B.18) \qquad \Delta_t \, \phi_{ij}^{A} = \eta \delta_j s_j + \gamma \, \Delta_{t-1} \, \phi_{ij}^{A} \quad \text{where} \quad \delta_j = \left(\sum_{k=1}^{K} \delta_k \phi_{jk}^{B} \right) f'(s_j)$$

■

C NOTE ON PROGRAMMING CODE

The computational analysis conducted in this paper was three-fold. First, the raw data had to be filtered. Secondly, our ANNs had to be constructed and lastly the $ARIMA$ models had to be estimated. The first two assignments were done by using *Visual Basic for Applications* (VBA). The last one was done using the statistical package R.

The data sorting algorithm was constructed using VBA forming a bridge between the database containing the raw unfiltered data and a spreadsheet with the filtered result. Our ANNs were made by using VBA in spreadsheets. This allowed us to tailor the network structure, learning algorithm and input details. This is not always possible with standard neural network software packages. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time series analysis, classification etc.) and graphical techniques. For computationally-intensive tasks, C and $C++$ code can be linked and called at run time. Advanced users can write C code to manipulate R objects directly.

R is a more diverse programming language and tailored for statistical analysis. We would therefore wish we had the time and resources to learn more of this language or some of the compatible ones mentioned above to be able to make a neural network model using R[12]. We would recommend future endeavors in neural networks to exploit the powerful R package with tailored programming. We highly recommend using more powerful language like $C++$ or $JAVA$ to build models from scratch instead of using VBA.

Computer codes in VBA for our data filtering algorithm and ANN algorithms can be obtained by mail, *s041732@stud.nhh.no*.

For more information on R, please see *http://www.r-project.org*.

[12] There are several user made packages on neural networks available for download, although none of these were suited for our analysis. With added customization, the network algorithms would have been faster and the model could have been more advanced with the time we had to our disposal.

REFERENCES

ADYA M. and COLLOPY F. 1998, *How Effective are Neural Networks at Forecasting and Prediction? A Review and Evaluation*, Journal of Forecasting, Vol. 17, pp. 481-495

AIT-SAHALIA Y. 1996, *Testing Continuous-Time Models Of The Spot Interest Rate*, Review Of Financial Studies, Oxford University Press For Society For Financial Studies, Volume 9, Number 2, pp. 385-426

AIT-SAHALIA Y. and MYKLAND P. 2003, *How Often to Sample a Continuous-Time Process in the Presence of Market Microstructure Noise*, Review of Financial Studies, Oxford University Press for Society for Financial Studies, Volume 18, Number 2, pp. 351-416

AKAIKE H. 1974, *A New Look At The Statistical Model Identification*, IEEE Transactions on Automatic Control, Volume 19, Number 6, pp. 716–723

ATIYA A. F. 2001, *Bankruptcy Prediction For Credit Risk Using Neural Networks: A Survey And New Results*, IEEE Transactions on Volume 12, Issue 4, pp. 929 – 935

BLUM A. 1992, *Neural Networks in C++*, New York, John Wiley & Sons

BOGER Z. and GUTERMAN H. 1997, *Knowledge extraction from artificial neural network models*, IEEE Systems, Man, and Cybernetics Conference, Orlando, FL, USA

BOLLERSLEV T., LITVINOVA J. and TAUCHEN G. 2006, *Leverage and volatility feedback effects in high-frequency data*, Journal of Financial Econometrics, Volume 4, pp. 353–384.

BOSARGE W. E. (1993), *Adaptive Processes To Exploit The Nonlinear Structure Of Financial Markets*, in TRIPPI R. R. and TURBAN E. (Eds.) *Neural Networks In Finance And Investing: Using Artificial Intelligence To Improve Real-World Performance*, Chicago, Probus, pp. 371- 402

BOX G. and JENKINS G. 1970, *Time Series Analysis: Forecasting And Control*, San Francisco, Holden-Day

CHAN M. C., WONG C. C. and LAM C. C. 2007, *Financial Time Series Forecasting By Neural Network Using Conjugate Gradient Learning Algorithm And Multiple Linear Regression Weight Initialization*, Computing in Economics and Finance 61

ENDERS E. 2005, *Applied Econometric Time Series*, 2nd edition, New York, John Wiley & Sons

FERNANDEZ A. and GOMEZ A. 2007, *Portfolio Selection Using Neural Networks*, Computers & Operations Research, Volume 34, pp. 1177-1191

GERLOW M.E., IRWIN S.H. and LIU T.R. 1993, *Economic Evaluation Of Commodity Price Forecasting Models*, International Journal Of Forecasting, Volume 9, pp. 387-397

HAGIWARA M. 1992, *Theoretical Derivation Of Momentum Term In Back-Propagation*, International Joint Conference on Neural Networks, Volume 1, pp. 682 – 686

HILL T., O'CONNOR M. and REMUS W. 1996, *Neural Network Models For Time Series Forecasts*, Management Science, Volume 42, Number 7, pp. 1082-1092

HORNIK K., STINCHCOMBE M. and WHITE H. 1989, *Multilayer Feedforward Networks Are Universal Approximators*, Neural Networks, Volume 2, Number 5, pp. 359-366

JORDÀ O. and MARCELLINO M. 2002, *Modeling High-Frequency Foreign Exchange Data Dynamics*, Macroeconomic Dynamics, Volume 7, pp.618-635

KAASTRA I. and BOYD M. 1996, *Designing A Neural Network For Forecasting Financial And Economic Time Series*, Neurocomputing, Volume 10, pp. 169-181

KLIMASAUSKAS C. 1993, *Making a Difference with Data Transformation*, Advanced Technology for Developers, High-Tech Communications, Sewickley, PA

KRISTENSEN T., TRECK B. and FALCK-OLSEN R. 1997, *Hypehenation By An Artificial Neural Network*, Norsk Informatikkonferanse NIK'97 Voss. Tapir forlag

KUAH K., BODRUZZAMAN M. and ZEIN-SABATTO S. 1994, *A Neural Network-Based Text Independent Voice Recognition System*, Southeastcon '94 'Creative Technology Transfer - A Global Affair', pp. 131-135

LEITCH G. and TANNER E. J. 1991, *Economic Forecast Evaluation: Profit Versus The Conventional Error Measures*, American Economic Review, Volume 81, pp. 580-590

MCADAM P. and MCNELIS P. 2005, *Forecasting Inflation With Thick Models And Neural Networks*, Economic Modeling, Elsevier, Volume 22, Number 5, pp. 848-867

MCCULLOCH W. S. and PITTS W. H. 1943, *A Logical Calculus Of The Ideas Immanent In Nervous Activity*, Bulletin of Mathematical Biophysics, pp. 115-133

MEESE R. and ROGOFF K. 1983, *The Out-Of-Sample Failure Of Empirical Exchange Rates: Sampling Error Or Misspecification?,* International Finance Discussion Papers 204, Board of Governors of the Federal Reserve System

MILLS T. C. 1990, *Nonlinear Time Series Models In Economics, Journal Of Economic Surveys*, Volume 5, pp. 215-241

MILLS T. C. and MARKELLOS R. N. 2008, *The Econometric Modeling Of Financial Time Series*, 3rd Edition, Cambridge University Press

MOALLEMI C. 1991, *Classifying Cells For Cancer Diagnosis Using Neural Networks*, IEEE Expert: Intelligent Systems And Their Applications, Volume 6, Issue 6, pp. 8-12

O'HARA M. 1995, *Market Microstructure Theory*, Oxford, Blackwell

OMRANE W. B. and OPPENS H. V. 2005, *The performance analysis of chart patterns: Monte Carlo simulation and evidence from the euro/dollar foreign exchange market*, Empirical Economics, Volume 30, Number 4, pp. 947-971

REFENES A. N., ZAPRANIS A. D., CONNOR J. T. and BUNN D. W. 1995, *Neural Networks In Investment Management*, TRELEAVEN P. and GOONATILAKE S. (Eds.) *In Intelligent Systems For Finance And Business*, John Wiley & Sons

ROSENBLATT F. 1958, *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain*, Psychological Review 65, pp. 386-408

RUMELHART D. E. and MCCLELLAND J. L. 1986, *Parallel Distributed Processing: Explorations In The Microstructure Of Cognition*, Cambridge MA, MIT Press

SAMARASINGHE S. 2007, *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*, Auerbach Publications

SCHWARZ G. E. 1978, *Estimating The Dimension Of A Model*, Annals of Statistics, Volume 6, Number 2, pp. 461–464

SEIJI H., TAKAHIRO M., YOSHIHIDE S. and TSUYOSHI N. 2004, *Neural Network Visual Inspection System With Human Collaborated Learning System*, IEEE ICIT '04, pp. 214-218

SHARDA R. and PATIL R. B. 1992, *Connectionist Approach To Time Series Prediction: An Empirical Test*, Journal of Intelligent Manufacturing 3, pp. 317-323

SHENG G., REIS M., CHURCHILL S. 2003, *A Zinc Finger Transcriptional Activator, Regulates The Transition Between Gastrulation And Neurulation*, Cell, Volume 115, pp. 603-613

SWINGLER K. 1996, *Applying Neural Networks: A Practical Guide*, London, Academic Press

TANG Z., ALMEIDA C. and FISHWICK P. A. 1991, *Time Series Forecasting Using Neural Networks Vs. Box-Jenkins Methodology*, Simulation 57, pp. 303-310

TAY A. and TING C. 2006, *Intraday Stock Prices, Volume, And Duration: A Nonparametric Conditional Density Analysis*, Empirical Economics, Volume 30, Number 4, pp. 827-842

THEIL H. 1966, *Applied Economic Forecasting*, Chicago, Rand McNally and Company

THIMM G. and FIESLER E. 1997, *High-Order and Multilayer Perceptron Initialization*, IEEE Transactions on Neural Networks, Volume 8, Number 2, pp. 249-259

TRIPPI R. R. and LEE J. K. 1996, *Artificial Intelligence in Finance and Investing*, McGraw-Hill

TSAY R. S. 2002, *Analysis Of Financial Time Series*, John Wiley & Sons

WHITE H. 1988, *Economic Prediction Using Neural Networks: The Case Of IBM Daily Stock Returns*, IEEE International Conference on Volume 2, pp. 451 - 458

ZHANG G., PATUWO B. E. and HU M. 1998, *Forecasting With Artificial Neural Networks: The State Of The Art*, International Journal of Forecasting 14, pp. 35-62

ZHANG G. 2003, *Time series forecasting using a hybrid ARIMA and neural network model*, Neurocomputing, Volume 50, pp. 159-175