Stock Price Prediction using Neural Networks

Master Thesis
Leiden University

F.W. Op 't Landt

4th August 1997

Contents

Chapter 1

Introduction

Stock price prediction is a rather hazardous operation. A good analyst is therefore not someone who is always right, but someone who is better at average, someone who has a higher efficiency than his colleagues.

In the last few years it has become clear that neural networks have become part of this class of analysts. Neural networks are programs that are based on the geometry of the human brain. The theory was developed in 1943, when the first computers were not even produced. The domain of neural networks has become one of the fastest growing sub-areas in computer science in the last ten years.

Neural networks are mostly good at recognizing complex patterns. A typical network receives large numbers of inputs and the expected outputs. It then searches for the relations between input and output. Once the computational rules have been found, the network is able to produce outputs on any input, but an error of a few percent is normal.

On the stock market a lot of information is produced in a short period of time. A fast response to this information is thus necessary. Most of the research is done in the area of the analysis of the time series (the prediction of future values based on stock history). However, the history is not the only factor of the stock to be predicted: the stock price development of for example Philips is highly dependent of the development of other electronics companies and of that of its own derivatives.

Macro-economic parameters on the other hand are much harder to process for short-term purposes. The cause is that these parameters are not regularly available. But in long-term management of portfolios, neural networks are very useful.

Time series analysis is used in many different areas [7]. Our objective in this particular case is to predict the next value in a time series: the next stock price.

1.1 Description of the project

This document contains the master thesis project, done by F.W. Op 't Landt under supervision of prof. dr. J.N. Kok of Leiden University and ir. M.N.

Hoevenaars of ING Nederland. The goal of this project is to develop neural networks, suited for stock price prediction, that is, to predict the stock price for a number of companies. The predictions will be made using feed-forward neural networks. The idea is to investigate whether feed-forward networks are able to make good predictions. The adaptations in the error measures and the kinds of networks will be considered and the results of these approaches will be compared.

1.2 Overview

We give an overview of the rest of the thesis. In Chapter 2 a general description of neural networks is given, followed by a discussion of some kinds of networks that are suitable for our problem area. In Chapter 3 the technique for developing a forecasting model will be given and the different steps that have to be followed will be discussed. In Chapter 4 the data set and the library used will be described. A few adaptations of the library had to be made: these adaptations will also be discussed. In Chapter 5 the data structures that are used in the program will be described. The data structures that we use are built by the library. In Chapter 6 the program for making the predictions is described. In Chapter 7 the strategies that we used for prediction will be discussed: the standard technique which uses only a companies own stock, and the leader/follower technique where prediction for the follower's stock is based on the stocks of other companies. In Chapter 8 the results will be given in which some statistics of the used techniques will be offered. In Chapter 9 a new approach will be discussed: we will discuss the possibilities and results of ensembles of neural networks. In Chapter 10 a number of possibilities for improvements will be given. Subjects for further investigation will be treated as well. The appendix contains the documentation of the program, including a brief user manual.

Chapter 2

Neural networks

In this chapter a description of neural networks will be given. In section 2.1 the adaptive behavior of neural networks and the computation of units will be described. Section 2.2 discusses a few types of networks that are suitable for our problem area.

2.1 Neural computation

The term neural computation refers to computation by artificial neural networks. The adjective "neural" indicates that the base of these networks lies in the field of neuroscience. Biological modeling however, is not of any concern. We want to use the abilities of *artificial* neural networks, which imitate the behavior of real neurons. If the term neural network is used, this refers to *artificial* neural networks (ANN's).

2.1.1 Adaptation of neural networks

An interesting property of neural networks is their ability to learn. Most newly programmed neural networks are not able to perform their task with the desired accuracy at once. Usually a network's behavior is adapted in a learning or training process. During this process the network is iteratively provided with a set of input patterns together with the corresponding output patterns until it produces the desired output. This set of input patterns and corresponding output patterns is called a training set. While training, the network may change the values of it's parameters according to the applied *learning rule*.

The purpose of training a neural network on a certain task depends on an important assumption. After the training phase the neural network is assumed to perform its task satisfactory on previously unencountered input patterns: the training is useful only if the knowledge gained from training patterns *generalizes* to other input patterns.

Therefore it is important for the training set to be representative for all input patterns on which the network will perform its task. Two conditions have to be fulfilled regarding the representativeness of training patterns:

- The training patterns must belong to the *class of patterns* which the network is expected to process. For example, if we want the network to predict the value of the next return index of ING, there is no use in training it on the interest rates.
- The training patterns must be selected from input space according to the *distribution* in which all input patterns occur in it. A network can not be expected to predict correctly when it is trained on a training set with too many outliers.

The network is usually trained to estimate the function (which is implicit in the training patterns) as closely as possible. A number of problems can arise which might prevent the network from learning this function:

- A satisfactory approximation can not be reached by a small network due to the lack of parameters to express the function.
- The network can suffer from overfitting to certain training examples. It will perform poorly on other input patterns. This problem can for example be solved by restricting the number of training cycles and the number of units in the hidden layer(s).
- Optimization techniques, where a minimization of an error- or cost-function is wanted, can become trapped in local minima, instead of propagating to the desired global minimum. Some of the most commonly used learning rules are believed to stabilize the network in local minima.

Many different types of neural networks have been investigated in the last few decades. These networks differ in areas such as used types of units, network topology, applied learning rule, and their behavior. For an overview consult [1]. We want to use feed-forward networks on the stock price prediction problem. The input patterns will contain a number of different stock values of a company to produce as output a future value of the company's stock price.

2.1.2 Computation of units

McCulloch and Pitts proposed in [1] a simple model of a neuron (see figure 2.1) as a binary threshold unit. Specifically, the neuron model computes a weighted sum $h_i = \sum_j w_{ij} V_j$ of its inputs from other units, and produces a one or a zero.

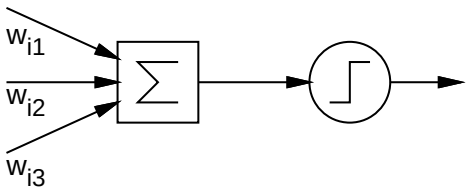

Figure 2.1: Model of the McCulloch-Pitts neuron

This depends on the value of the sum: whether it is above or below a certain threshold. The networks in this project are built from these neurons, which are connected through unidirectional links. Each link has a value, the weight of the link. Each unit produces a continuous valued activation V_i. The activation of a unit is

$$V_i = f(\sum_j w_{ij} V_j - \theta_i) = f(h_i - \theta_i)$$

where $f(h)$ denotes the activation or transfer function. We will omit θ_i because this threshold value can be simulated by a link from a fixed-value unit.

Now we can make a classification of units based on their transfer function:

- Threshold units incorporate a threshold function. The activation functions are restricted to 1 or 0, as stated before. For optimization techniques on an error- or cost-function a continuous differentiable transfer function is desirable. These units have continuous valued activations.
- Linear units use a function $f(h) = h$.
- Non-linear units are most commonly used in gradient descent learning. These units mostly have sigmoid transfer functions, such as $f(h) = tanh(h)$ or $f(h) = \frac{1}{1+e^{-h}}$.

The three different transfer functions are shown in figure 2.2

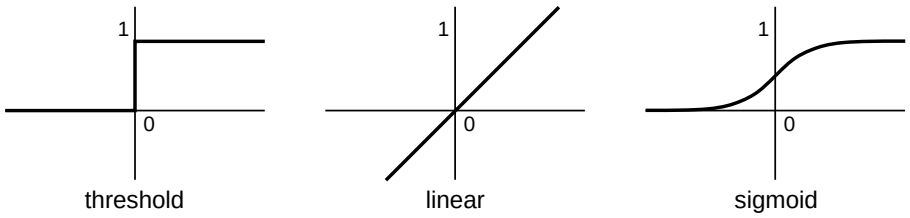

Figure 2.2: Three different transfer functions

2.2 Feed-forward networks (FFNs)

In this section we will describe some neural network methods for time series analysis. The section is structured as follows. In a number of sections we discuss different approaches based on neural networks.

First the possibilities of feed-forward networks for time series analysis will be discussed. Then the properties of recurrent networks will be explored. Recurrent networks resemble layered feed-forward networks and are also suitable for the use on time series. We will discuss this approach in sections 2.2.5 and 2.2.6. The properties of ensembles of networks will be discussed in chapter 9.

2.2.1 Feed-Forward networks with time-windows

Layered feed-forward networks are often called perceptrons. Figure 2.3 shows two typical examples of perceptrons. There is a set of input terminals whose only role is to feed input patterns into the rest of the network. After this layer there are one or more intermediate layers of units, followed by a final output layer from which the result of the computation is read. Every unit feeds only the units in the next layer.

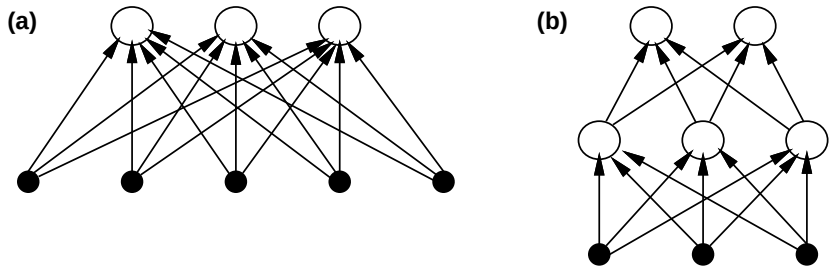

Figure 2.3: Perceptrons. (a) A simple perceptron which (by definition) has only one layer. (b) A two-layer perceptron.

The function g is the activation function of the units. The function g is usually nonlinear. The main advantage of nonlinear activation functions is that they can keep the output between fixed bounds. Multi-layer networks can solve problems, using these nonlinear functions, which they can not solve when they use linear units. The output is an explicit function of the input; the input is propagated through the network and produces the output right away. Often continuous-valued units (with g a continuous and *differentiable* function) are considered. It is possible to construct a cost function $E[\mathbf{w}]$ which measures the system's performance error as a differentiable function of the weights $\mathbf{w} = \{w_{ik}\}$. We can then use various optimization techniques, such as gradient descent, to minimize this error measure.

Feed-forward networks can be used for the prediction of the next value in a time series. The idea is to offer not only the present input, but also $N-1$ previous inputs to the network, together called a "time window". In a number of applications good results have been obtained with this kind of networks [1]. A drawback is that the number of input units is proportional to N, resulting in extra weights to be updated during training. Another disadvantage is the fixed value of N. The performance may depend heavily on the choice of the value of N. We have decided to experiment with a number of different values for N.

We have experimented in addition with the error measure. It is also possible to preprocess the input. These are the topics of the next two subsections.

Different error measures

Learning is based on an error measure. The idea is that the smaller the error measure or cost function is, the better the weights w_{ik} of the network are.

Given the error measure function $E[\mathbf{w}]$, we can improve on a specific set of weights (w_{ik}) by sliding down hill on the surface it defines in $\mathbf{w}$ space. Specifically, the usual gradient descent algorithm suggests changing each w_{ik} by an amount Δw_{ik} inverse proportional to the gradient of E at the present location.

Often the mean squared error is used, but there are many alternatives. If we take N to be the number of samples in the data set, the mean squared error (MSE) is defined by:

$$MSE = \frac{1}{N}\sum_{\mu=1}^{N}(O^{\mu} - T^{\mu})^2,$$

where O is the produced output and T is the expected output.
If training with the standard error measure, described above, turns out to be too slow or not adequate, alternative error measures can be used. Examples include the prediction of change in direction error, or POCID error. If the target value is greater than the previous target value in the series, then the output value must be at least equal to the previous output (and vice versa; see figure 2.4). The POCID error is the total number of errors relative to the total number of predictions N.

$$POCID = \frac{1}{N}\sum_{t=1}^{N} D_t$$

where

$$D_t = \begin{cases} 1 & if\ (tar(t) - tar(t-1)) \cdot (o(t) - o(t-1)) \ < \ 0 \\ 0 & otherwise \end{cases}$$

The POCID lies always between 0 and 1.
The goal is to minimize the POCID error.

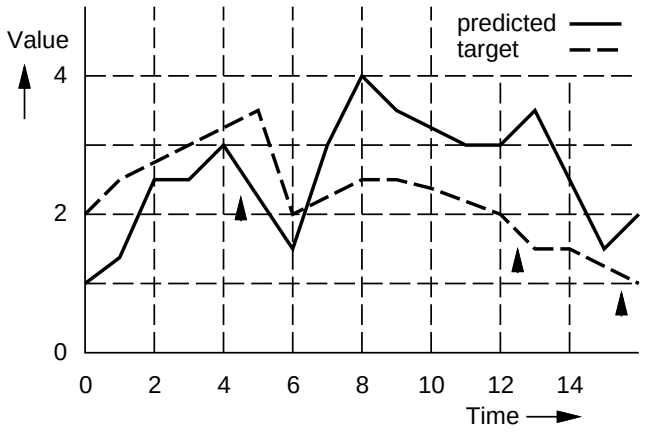

Figure 2.4: Prediction Of Change In Direction. A POCID error occurs if the target and the predicted value change in opposite directions (one increases, the other decreases). The errors are marked by the arrows.

The error described above can be used to train the network. Only if an output changes in the opposite direction of the target ($D_i = 1$), the entire error is used to calculate the weight change. Otherwise, only a fraction of the error is back-propagated. We propose to experiment with the different error criteria, in particular with the POCID error.

Preprocessing

Another possibility for improving the feed-forward network approach is preprocessing of the input data. We can filter the input data in many different ways, for example using Wavelet transformations or taking the local averages of the data using different Gaussian functions, for smoothing the data.

It is also possible to train the parameters of the filters. In this way we can try to find optimal filters for the data, and the best approximations of the wanted output.

2.2.2 Radial basis networks

Recently, variations on feed-forward networks have been proposed [13]. A radial basis network is a feed-forward network that uses a different kind of transfer function in the first hidden layer (e.g. Gaussian functions). In this way a network is better able to explore features. Radial basis networks are similar to feed-forward networks, but radial basis networks train more rapidly, while exhibiting not so much of back-propagation's training pathologies such as local minima problems [1]. They have one major disadvantage however: after training they are generally slower to use, requiring more computations to perform a classification approximation.

2.2.3 Network construction algorithms

A different approach is to have "self-building" neural networks. This type of network constructs its own architecture while training the new added units on the input data. The Group Method of Data Handling (GMDH) [12], [5] belongs to this category, and has already been applied to the stock price prediction problem by the ING. Two alternatives can be investigated: Cascade-correlation and non-linear lower dimensional representation (NLDR) networks.

Cascade-correlation is a supervised learning algorithm which builds a neural network as part of the learning process. The algorithm consists of the iteration of two phases: output unit training and hidden unit training.

The output units receive input from all input and hidden units. In the first phase, output unit weights are trained to minimize the sum squared error measure. Once this training process levels off, a final epoch is run to record the residual error for each unit on each training pattern. If the error remains above a certain threshold, a new hidden unit is inserted.

The new hidden unit receives input from the input units and all previously created hidden units. Thus, the hidden units form a cascade. The new hidden unit weight is trained to maximize the sum of the magnitude of the correlation

between output values for each pattern and the residual error from the previous output training phase. See figure 2.5 for an example.

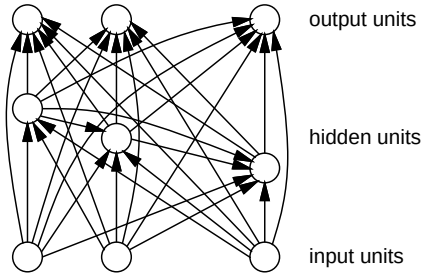

Figure 2.5: Example of cascade-correlation

NLDR networks use a technique for recoding multidimensional data in a representation of reduced dimensionality. For scalar time series data, a common technique is phase-space reconstruction by embedding the time-lacked scalar signal in a higher dimensional space. The idea is to reduce the dimensionality of a set of input data in a non-linear way. By using this reduction, the correlation between the data is also reduced. See figure 2.6 for an idea of how this network works.

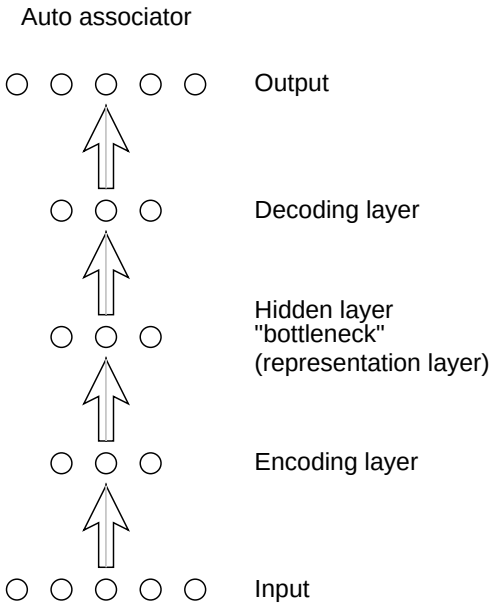

Figure 2.6: A network capable of non-linear lower dimensional representations of data

2.2.4 Sub networks

In a number of applications we have to deal with heterogenity: that is, we need different subnetworks for certain parts of the data. It is a good idea to investigate whether this is also the case for the stock time series by using feed-forward networks in combination with the Least Vector Quantization (LVQ) algorithm of Kohonen. The LVQ algorithm clusters the input data, and sends it to the appropriate subnetwork.

2.2.5 Simple recurrent networks (SRNs)

The topology of SRNs is similar to that of a layered feed-forward networks, but there are additional feedback connections that make the network suitable for the use on time series. A SRN typically consists of one input, one context, one hidden, and one output layer. The context layer receives feedback from a higher layer (i.e. closer to the output layer) or from itself and it provides the network with information about previous activations of certain units. However, the weights are set to fixed values, and only the weights on the forward connections are trained using for example the standard back-propagation algorithm. We plan to investigate whether SRNs and which type of SRNs can be used for our purposes.

There are many different architectures possible for this kind of networks, and we can use genetic algorithms to find it. This genetic algorithm can train and test different networks in parallel on the nCUBE supercomputer and select the best one for the stock price prediction problem.

2.2.6 Fully recurrent networks

The difference between SRN's and fully recurrent networks is that a fully recurrent network includes direct or indirect loops of connections, and the weights on these connections can be learned. Examples of such networks are Hopfield and Boltzman networks. Training can be difficult, and time consuming. However, if the feed-forward networks do not give the desired results, this might be a good alternative.

2.2.7 Remark

For most of the proposed alternatives of the feed-forward network the core of the code can be re-used. The adjustments between the different approaches on the level of code are not too big.

2.2.8 Consideration

In this project we will not try all of the proposed kinds of networks. FFNs with time-windows will be used for our problem. We will experiment with the different error measures (MSE and POCID) and the size of the learning set.

Chapter 3

Developing a forecasting model

There is a large amount of interest for the use of neural networks in the domain of stock price prediction [6], [11], [12]. The neural networks can be retrained from time to time with the latest data, for example including earnings results and interest rates.

3.1 Development of a neural network

There are many steps in building a forecasting model, as stated in [11]:

1. Decide on what your target is and develop a neural network (following these steps) for each target.
2. Determine the time frame that you wish to forecast.
3. Gather information about the problem domain.
4. Gather the needed data and get a feel for each inputs relationship to the target.
5. Process the data to highlight features for the network to discern.
6. Transform the data as appropriate.
7. Scale and bias the data for the network as needed.
8. Reduce the dimensionality of the input data as much as possible.
9. Design a network architecture (topology, number of layers, size of layers, parameters, learning paradigm).
10. Go through the train/test/redesign loop for a network.
11. Eliminate correlated inputs as much as possible, while in step 10.
12. Deploy the network on new data and test it and refine it as necessary.

In the next subsections some of the steps mentioned above will be examined in more detail.

3.1.1 The target and time frame

First we have to choose the target. The target of our neural network is to predict values of the stocks. Another possibility is to predict the direction of a certain index, if we want to give an evaluation for a number of companies. In that case the direction of the company's index is important, not the exact price index.

Another decision has to be made: the size of the time frame. A neural network model for a short-term prediction is harder to create than for a longer term prediction. The seemingly random, chaotic variations at smaller time scales, and the appearance of market noise might explain this. The macro economic forces that, on the other hand, fundamentally move market over long periods, move slowly. For a given error tolerance, a one-year forecast, or one-month forecast will take less effort with a neural network than a one-day forecast will.

For the given problem a short-term prediction will be used. However, for an investment the expectations in the further future are more important than those of the near future, so then a long-term prediction should be used. The adaptations that have to be made to prepare a short-term prediction trained FFN for long-term prediction are expected to be minimal. It is quite likely that only the number of input and hidden nodes have to be redetermined.

3.1.2 Domain expertise

To build an effective predictive model of the stock market, or another financial market, knowledge about the factors that influence the market is required. So we have to investigate this domain before training the network and retrieving data.

3.1.3 Gathering data

The data to be used for training, testing and running the network, will be retrieved from DataStream. DataStream is a provider of stock data for ING. This provider can offer us a lot of different kinds of data, such as return indices, interest rates, bonds, etc. of a large number of different markets.

3.1.4 Preprocessing the data for the network

It is possible to preprocess the data before giving it to the neural network. For example we can use a Wavelet transform to filter out the correlations between the data, as discussed in chapter 2. This is a rather difficult operation, and if the network performs good on scaled data, then this is often not necessary.

3.1.5 Finding features in the input data

A possibility is to try to find leaders and followers in the stock market. The values of some companies can then be used to predict a value for some other company. For example: there seem to be 18 leaders for the Dutch fund of

Philips (from [14]). These leaders seem to determine the future value of Philips by some timeshift. We can use the values of the leaders to predict the stock of Philips.

3.1.6 Transform the data if appropriate

Besides transformation of the data, it is also possible to scale the data. This option will be discussed in the next subsection.

3.1.7 Scaling data

The scaling operation which is used by our networks, is implemented as follows: while loading the data from an input file, the minimum value, maximum value, mean value, number of values, and standard deviation are determined. First, the mean value is computed: add each value that is read to `mean` and divide by the number of values. If N is the size of the data set, then the standard deviation is computed by:

$$sd = \frac{1}{N} \sum_{\forall\, x\, \epsilon\, dataset} \mid x - mean \mid .$$

Neurons like to receive the input data in a certain input range to be most effective. Input data which vary between 3 and 110, like the stock price values of Ahold, will not be useful, since the neurons in the hidden layer have a sigmoid activation function that squahses large input values to either 0 or +1 (see figure 2.2). So we should choose data within a range that does not saturate, or overwhelm the neurons. A good idea is to choose inputs from -1 to 1. We can scale the inputs using the following function:

$$x' = \frac{x - \overline{x}}{sd},$$

that is, subtract the average and divide by the standard deviation.

Scaling of the outputs:

$$x' = \frac{x - min}{max - min}.$$

After scaling the inputs will be situated around zero (even rather close to zero), and the outputs are guaranteed to be between zero and one. This is exactly what we wanted, because the sigmoid function, which is used for the activation of the units, is bounded by zero and one (see figure 2.2). After the output has been produced, it will be scaled back to the actual value (i.e. the actual prediction value):

$$x = (x' \cdot (max - min)) + min.$$

3.1.8 The train/test/redesign loop

Most of the process of determining the best parameters is trial and error. The different options to find the best fit for the problem have to be explored. The following steps have to be taken:

Split the data. Divide the data set into a learning set and a testing set. Use about 80% for the training set, and 20% for the testing set.

Train and test. Start with a network topology and train the network on the training data. When a satisfactory minimum training error is reached, apply the trained network on the test data and note the error. Restart the process with the same network topology for a different set of initial weights and see if a better error on training and test sets can be reached. The reason is that the network may have found a local minimum on the first attempt and randomizing initial weights may lead to a different, maybe better solution.

Eliminate inputs. Try to reduce the number of inputs, by iteratively removing an input and noting the best error that can be achieved on the training and test sets.

Iteratively train and test. Repeat the train and test process to achieve a better result.

Deploy the network. Use the test set to see how the optimized network performs. If the error is not satisfactory, the design phase or the train and test phase has to be re-entered.

Revisit the network when conditions change. The network has to be retrained, when there is reason to think that new information (relevant to the problem) can improve the performance of the network. The FFN for the stock price prediction problem should be retrained at least once a week, if not on a daily basis. If the network seems no longer to generalize well with the new information, the design phase has to be re-entered.

Taking these steps, the following parameters have been found to be best for the FFN designed for our stock price prediction problem (prediction of one day ahead for Ahold):

number of input units	10
number of hidden units	4
threshold	0.1 or less
eta	1.0
alfa	0.9
randomized starting weights	$[-0.01,\ 0.01]$
final MSE	0.01

Chapter 4

Data set

In this chapter the data set is discussed. We will introduce the software and the data sets. Then a discussion about the size, format and conversion of the data set will be given. Possibly an adaptation of the code of the library may have to be made; this is the topic of section 4.5.

4.1 Software

The program that we wrote offers the possibility to build FFNs with a variable number of input and hidden nodes. There is always one output node, as we only expect one value for prediction. Furthermore, one can choose the error threshold, learning rate, etc. For the implementation of the FFN the SPRANNLIB is used [4]. This is a library which contains standard operations on neural networks. It was developed by the Pattern Recognition Group of Delft University of Technology. The library uses the Numerical Recipes in C [10], which we obtained from Harvard.

4.2 Size of the data set

The data sets will be retrieved from the DataStream provider. ING ITResearch has a dish antenna which provides real-time data, that for security reasons, are presented for test purposes with a delay of 15 minutes. However, we will use the closing price of the funds only. Old data sets are fed into the FFN for training, while for the prediction of future values the FFN is provided with new data.

The size of the data sets for training can extend to up to 100 Mb. This could cause memory problems, as the SPRANNLIB loads complete data sets from files into a pointer structure, in stead of handing parts of it to the FFN.

An earlier program for stock price prediction [12] at ING loaded the data sets from an Oracle database. In this case the use of embedded SQL is needed, because the rest of the program is written in C. The data sets reside on the nCUBE supercomputer, where they can be retrieved by the program. The nCUBE is a parallel computer, where a program can be run on multiple processors (always a power of 2).

We would like to use a similar technique. The problem is that the data have to be read from files. In our solution the data are placed in an array, from which the samples are made. In stead of using a pointer structure, the samples are stored in an array structure as well. This adaptation has increased the (time) performance considerably.

4.3 Format of the data set

Each line of the data set must contain a date between quotation marks, followed by a value or `NA` (`N`ot `A`vailable). The date and the value are separated by a comma, and only one day per line is allowed. The date has the following format: `"dd/mm/yy"` or `"dd-mm-yy"`. That is, two positions for the day, two positions for the month, and two positions for the year. The value may contain a point, but never a comma. The program demands that all lines contain subsequent dates, except for the weekends which may be skipped. This means that each line has the following format:

`"dd/mm/yy"`	this day's value	The date and value of the index
	OR	
`"dd-mm-yy"`	this day's value	The date and value of the index

The data set may be preceded by any form of comment, provided that the second position of the line is not a number.

The format of the data sets which are provided by DataStream is appropriate for our program. The data sets provided by DataStream have the following format:

`"Name"`	name of enterprise	For recognizing the company
`"Code"`	code for enterprise	The code for retrieving the data
`"Currency"`	currency	The currency of the values
`"dd/mm/yy"`	this day's value	The date and value of the index

An example data set of DataStream:

```
"Name","ING CERTS."
"Code",531865
"Currency","FL"
"28/02/91",NA
"01/03/91",NA
"04/03/91",46.75
"05/03/91",47.84
"06/03/91",48.73
"07/03/91",48.24
"08/03/91",48.04
"11/03/91",46.95
"12/03/91",46.75
"13/03/91",46.46
"14/03/91",47.44
"15/03/91",47.84
```

If the value for a day is `NA`, this means that for this day the value was `N`ot `A`vailable. Furthermore each weekend is skipped, and the data are simply given until the last given date is reached.

4.4 Conversion to SPRANNLIB format

The library (SPRANNLIB) offers the possibility to convert data to SPRANNLIB format. This function reads the data from one file, converts them to SPRANNLIB format, and puts them into another. A number of options can be chosen to indicate the number of input and output values to create, the type of data set, etc.

This is another obstacle, because, considered the huge size of the data sets, it adds a large amount of time to the time needed to train the FFN.

We have decided not to use this function, but to write an alternative function instead. This function offers the possibility to use delays of fixed size between the input data, and a delay between the last input value and the value to be predicted. The function fills an array with samples, which are all in SPRANNLIB format.

4.5 Adaptation of SPRANNLIB

SPRANNLIB loads the complete data set into a pointer structure. This may cause problems with the internal memory, so an adaptation of the code is desirable. The adaptation of the code involves an extensive investigation of the library: all uses of the pointer structure have to be replaced. As described above, a new function has been written, and certain adaptations haven been made. Most of the standard SPRANNLIB functions could be used, so the adaptations are limited.

4.6 Training on the data set

Once we have generated a correctly classified and well distributed training set, we have to offer the samples in random order. If this is not done in random order, the network is trained initially on the first obtained stock price values and it will forget these examples further on in the training session. By randomizing the order of samples, the network will learn values from the first period together with later samples. In this way we try to keep the network from forgetting formerly learned features.

4.7 Generalization and memorization

We want the FFN to memorize the offered training samples. In order to check if the network is not overtrained, a number of test samples is kept aside from the training set. With these test samples we are able to see whether the network is able to predict correctly. If the network responds poorly to the test set, we know that we have overtrained. Otherwise we can say that the network *memorized*

the training patterns. In figure 4.1 an arbitrary curve-fitting analogy is shown. The generalized fit is labeled G, and the overfit is labeled O. In the case of the overfit any data point outside of the training data results in a highly erroneous prediction.

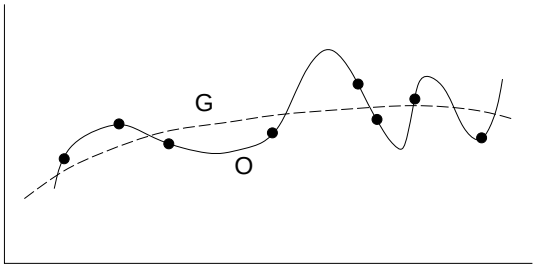

Figure 4.1: General (G) versus overfitting (O) of data.

To prevent the network from overfitting, the number of inputs should be reduced. The objective is to find the function with the least inputs that fits the data adequately. We have to be careful with having too many (unimportant) inputs: the results of the training data may be very good, but the network can perform extremely poor on the test data.

Chapter 5

Data structures

We use an adapted version of the data structure of the SPRANNLIB [4]. A summary of the ideas of that paper is given in this chapter. For a more detailed treatment of the data structures, see [4]. Furthermore, the used data structure for training, testing and running the network is not apt to the stock price prediction model. This is caused by the fact that the data sets that ING offers are much too big to be placed in a file. Another problem that was encountered, is the fact that the SPRANNLIB has no functions to handle delay between inputs, or to predict some days, or even more, ahead. Therefore we rewrote this part of the code for our application. The idea is to upload the data on-line and offer them directly to the feed-forward network, that is built using functions of the SPRANNLIB.

5.1 The top level data structure

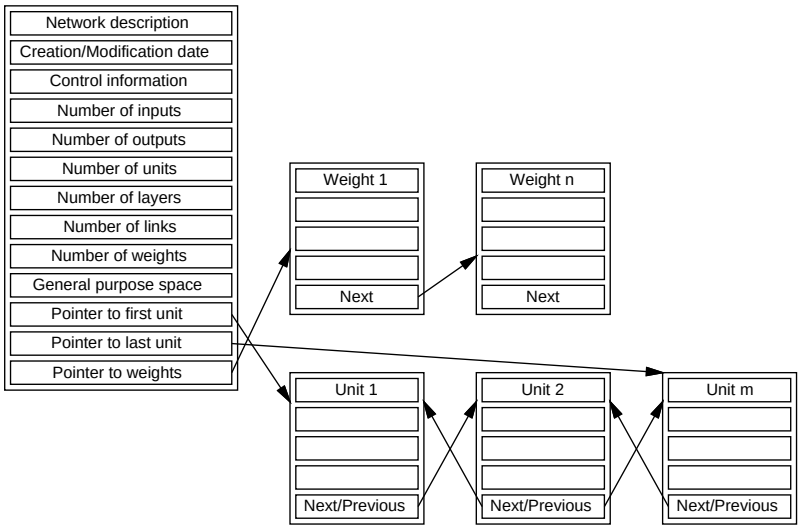

Figure 5.1: The top network data structure, from SPRANNLIB

The top of the network data structure (see figure 5.1) is a network header. All network related functions use a pointer to this data structure on input and all variables can be accessed from this structure.

A pointer to the first and last unit of the double linked list of units gives access to the whole network structure. A pointer to a linked list of all weights is provided for quick access to the weights.

5.2 The unit structure

The units (see figure 5.2) together with the links determine the network topology. The units are arranged in a double linked list. A linked list of links is connected to each unit. This list specifies the units or input terminals which are inputs to the units. A second linked list of links determines the units to which the output is transferred.

A pointer to the unit activation function and units transfer function allows different types of units to be mixed in the same network. The user can decide on which function to use. The transfer functions are implemented in a similar way.

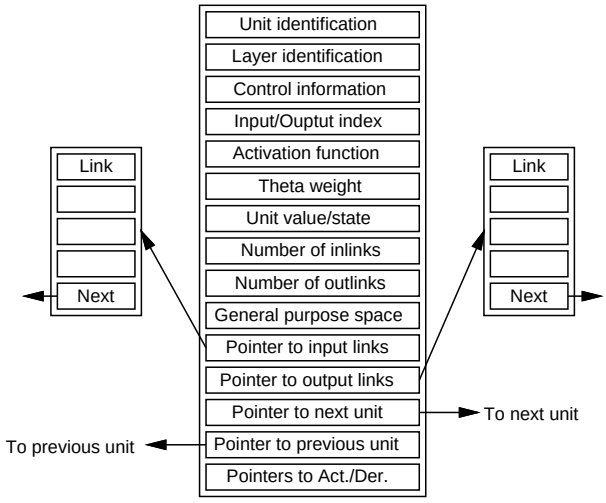

Figure 5.2: The unit data structure, from SPRANNLIB

5.3 The link data structure

A link data structure (see figure 5.3) represents a connection from one processing unit to another unit in a network. Each unit contains input links: links associated with the connections coming to the units, and output links: links that transfer the unit output to other units. Each link also contains a pointer to a weight structure, holding the connection strength.

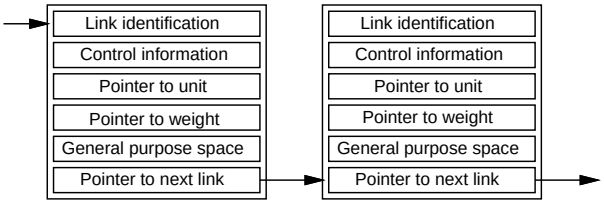

Figure 5.3: The link data structure, from SPRANNLIB

5.4 The weight data structure

Weights are arranged in a single linked list of weight structures (see figure 5.4). The weight structure also holds a pointer to a list of previous weight values. This makes it possible to store the history of the network for studying the evolution of the weights during training.

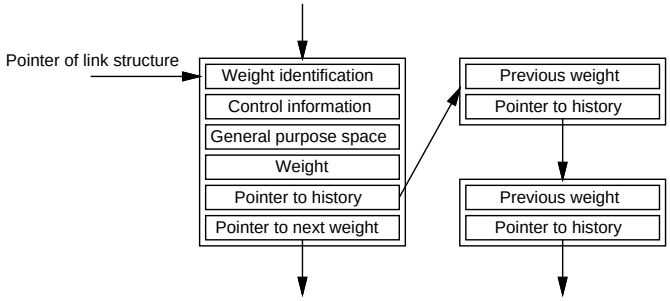

Figure 5.4: The weight structure, from SPRANNLIB

5.5 The unit value structure

The unit value (see figure 5.5) holds the current activation. This structure holds, among others, the current error, which is used for back-propagation. A pointer to a history is found here too. This allows the possibility to record a history of activations.

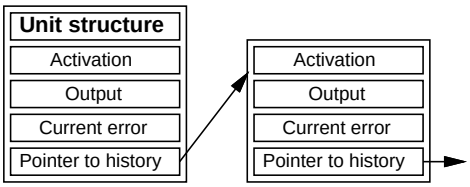

Figure 5.5: The unit value and history values, from SPRANNLIB

5.6 An example network

An example of the implementation of a 4-2-1 network is given in figure 5.6. In the example all relevant pointers are shown. These are: pointers from weights to weights, pointers from unit to unit (except pointers between units in different layers to maintain overhead), pointers from links to links, pointers from links to weights, pointers from links to units and the top network structure (Net) with pointers to the first and last unit.

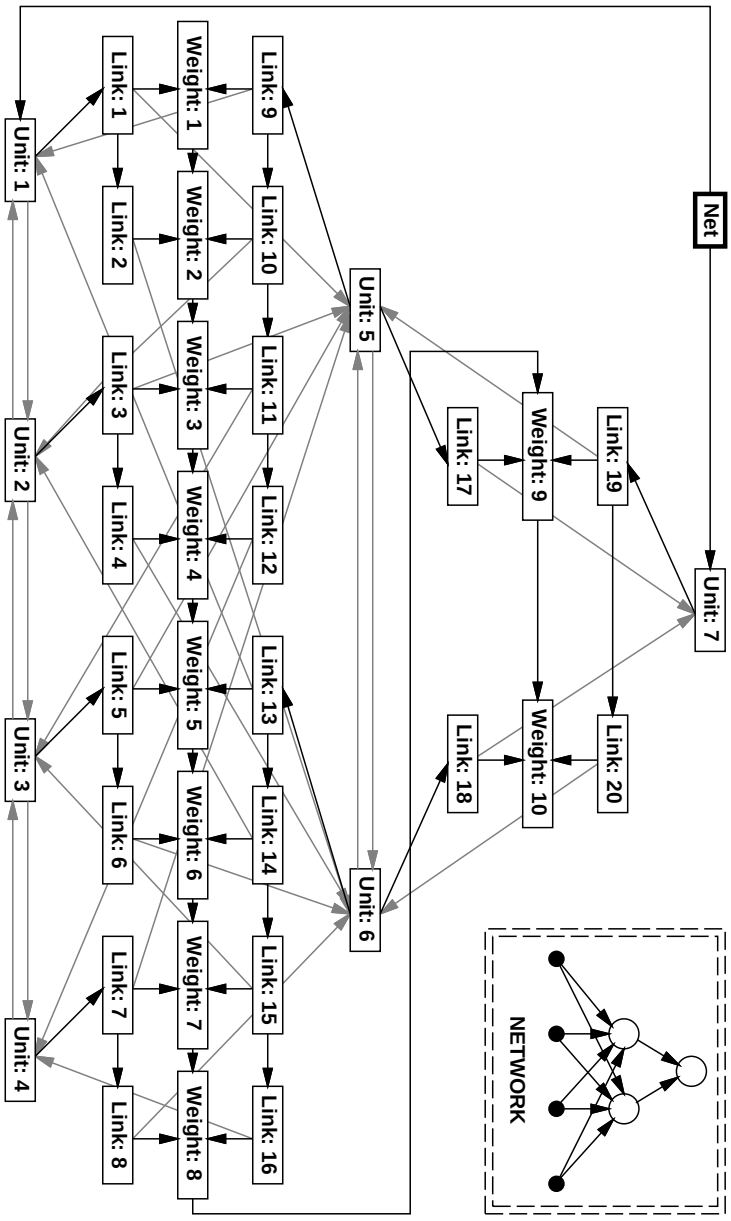

Figure 5.6: Example of a 4-2-1 network

Chapter 6

The program

In this chapter the implementation and functioning of the sequential program will be described. It will be a global description, as special attention will be paid to the different parameters of the program in the appendix.

6.1 Implementation

For implementation of the feed-forward network a pointer structure is used. This structure is built by the SPRANNLIB. The data structures are already discussed in chapter 5. In this chapter it was also mentioned that a new function had to be built for constructing samples. This new function makes a delay between input values possible, as well as (in the learning stage) prediction of a certain number of days ahead.

The program builds a list of learning and a list of testing samples, which will be offered to the network. The list of learning samples contains the samples in random order to keep the network from forgetting formerly learned features (as discussed in chapter 4). The list of testing samples on the other hand are in time sequence. That is, in the order in which they appear in time.

6.2 The general program

The program offers a number of options to the user, which the user can provide in a batch file too. After starting up the program, it will perform according to the wishes of the user. Now a full run of the program will be described, going through all main features of the program:

- Load the learning data into a table and make a list of samples according to the desired delay and prediction time.
- Load the testing data into another table and make a list of samples using the same delay and prediction time.
- Measure the initial performance of the learning and testing set.

- Perform back-propagation using the mean squared error as stop criterion for learning, while never exceeding the maximum number of cycles, or perform back-propagation using a fixed number of cycles as stop criterion.
- Measure the final performance of the learning and testing set.
- Evaluate the testing set. In this function the calculated values are written to an output file, which thus contains the actual prediction on the testing set.
- Compute the execution time of the program.

When the program is used for prediction on new values, the network should be trained up to the date to be predicted, before making a forecast. A new learning set has to be made, which contains the values up to the desired date.

It is also possible to make an entirely new prediction (i.e. a prediction where the target is unknown). In this case a different sample should be made, namely one that contains the inputs only. The testing samples contain the target value, but for a totally new prediction the value is of course unknown. When a completely new value has to be predicted, the network that will generate the prediction is assumed to be trained on the same delay between the inputs and the same time ahead as the prediction to be made has.

Chapter 7

Strategies for prediction

Many different strategies can be used for stock price prediction. We have tried two of them. When a prediction of a certain company is desired, one of the following options can be chosen:

- Stock price prediction using a FFN. This will be called the standard technique.
- Stock price prediction using the leader/follower technique. The idea is to give the FFN the leaders as inputs, to produce the value for the follower.

Further options can be provided by training the network on different error criteria, for example the POCID, which includes direction into the error criterion. These options will be discussed in the following sections.

We have run the program using the stockdata of Ahold. Therefore, the test results and used parameters are only representative for this particular company. In this chapter only a few test results are given to illustrate the different techniques. We will elaborate on the results in chapter 8.

7.1 Standard technique

The standard technique offers a number of inputs to the FFN. These inputs are values from one company which are situated within a certain time-window. For example, we can take 10 values of the company's history, each with the same delay in days between one and the following value. These values are processed through the network to produce a certain output, using the back-propagation algorithm. In the training phase, a target output will be provided. Based on the difference between the computed and the expected output, the weights of the network will be adapted. In this way, a diagram can be produced which resembles the actual stock. An example diagram is shown in figure 7.1.

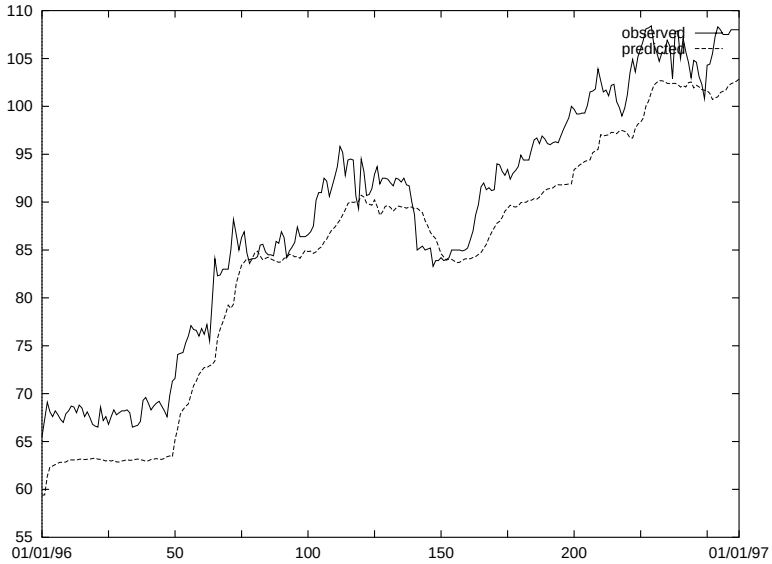

Figure 7.1: The results on data not seen before of the standard technique trained on the MSE and the actual stock.

7.1.1 Mean squared error

The results that we obtained after training the network on the MSE can be seen in the next table. The first column indicates the network number. This number is arbitrary. The next columns give the MSE, normalized MSE, and the percentage of correct predicted directions for each network. These results were obtained by simulating one year's predictions. The stock values of weekdays in the period of 01/01/96-01/01/97 are given to the network as inputs. We used a training set of 40 samples. After producing an output value, the window is moved one day and the network is retrained on the updated training set.

network	MSE	normalized MSE	correct direction
01	21.82	0.0119	46.01%
02	22.11	0.0120	45.25%
03	21.71	0.0118	45.25%
04	25.20	0.0137	46.39%
05	21.59	0.0117	46.77%
06	25.10	0.0136	47.15%
07	21.60	0.0117	45.25%
08	20.45	0.0111	46.77%
09	19.36	0.0105	45.63%
10	24.01	0.0130	48.29%
average	22.30	0.0121	46.28%

7.1.2 Prediction of change in direction

The results that we obtained after training the network on the POCID can be seen in the next table. The first column indicates the network number. This

number is arbitrary. The next columns give the MSE, normalized MSE, and the percentage of correct predicted directions for each network. These results were obtained by simulating one year's predictions. The stock values of Ahold of weekdays in the period of 01/01/96-01/01/97 are given to the network as inputs. We used a training set of 40 samples. After producing an output value, the window is moved one day and the network is retrained on the updated training set.

network	MSE	normalized MSE	correct direction
01	16.86	0.0092	64.26%
02	20.22	0.0110	68.06%
03	19.35	0.0105	64.26%
04	23.55	0.0128	68.44%
05	20.73	0.0113	65.78%
06	25.76	0.0140	65.40%
07	23.70	0.0129	67.30%
08	27.30	0.0148	66.54%
09	21.62	0.0117	65.02%
10	24.36	0.0132	60.84%
average	22.35	0.0121	65.59%

The values obtained by training the network with the POCID error can be seen in figure 7.2. In this approach the direction of the stock is considered more important than the precise value. We combined the two approaches in order to keep the prediction close to the actual stock.

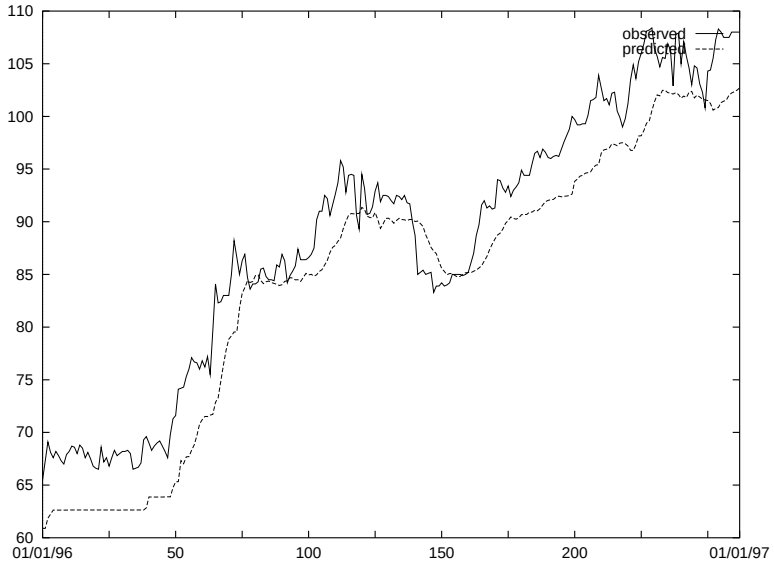

Figure 7.2: The results on data not seen before of the standard technique trained on the POCID error and the actual stock.

7.1.3 Conclusion

The standard technique using the MSE as error measure for training, produces slightly better results if we consider the MSE. The standard technique using the POCID error measure for training however, performs much better if we consider the percentage of correct predicted direction. Since the direction of the stock is more important than the exact value of the stock, training on the POCID is the best choice.

7.2 Leader/follower technique

The leader/follower technique is based on the idea that the stock of some companies may influence the stock of others. ING did some research on this subject, and they have found a number of dependencies. The stock used in this document is the stock of Ahold. Ahold has been found to be a follower of 30 leaders in many different areas. Each leader causes an effect in the stock of Ahold. Let's assume that some leader has an increase/decrease in its stock some day, then the stock of Ahold will respond with an increase/decrease some days later. The number of days between the symptom in the stock of the leader and the impact on the stock of Ahold is taken as the delay for that leader.

The target value of Ahold is chosen from the Ahold stock history, and the inputs are taken from the stocks of the leaders accordingly. In this way, the FFN is trained to react to increases and decreases in the stocks of the leaders of the fund it is trained on. The rise and fall of the fund will be predicted, based on the stocks of the leaders.

7.2.1 Mean squared error

The results that we obtained after training the network on the MSE can be seen in the next table. The first column indicates the network number. This number is arbitrary. The next columns give the MSE, normalized MSE, and the percentage of correct predicted directions for each network. These results were obtained by simulating one year's predictions. The stock values of Ahold of weekdays in the period of 01/01/96-01/01/97 are given to the network as inputs. We used a training set of 40 samples. After producing an output value, the window is moved one day and the network is retrained on the updated training set.

network	MSE	normalized MSE	correct direction
01	22.09	0.0120	69.47%
02	20.87	0.0113	66.41%
03	26.14	0.0142	66.03%
04	23.85	0.0130	69.08%
05	23.73	0.0129	64.89%
06	21.27	0.0116	67.56%
07	24.45	0.0133	68.70%
08	26.51	0.0144	66.41%
09	21.57	0.0117	69.08%
10	26.06	0.0142	69.47%
average	23.65	0.0129	67.71%

The outputs of the network should be as close as possible to the target values. Therefore the network is trained on the MSE. An example diagram is shown in figure 7.3.

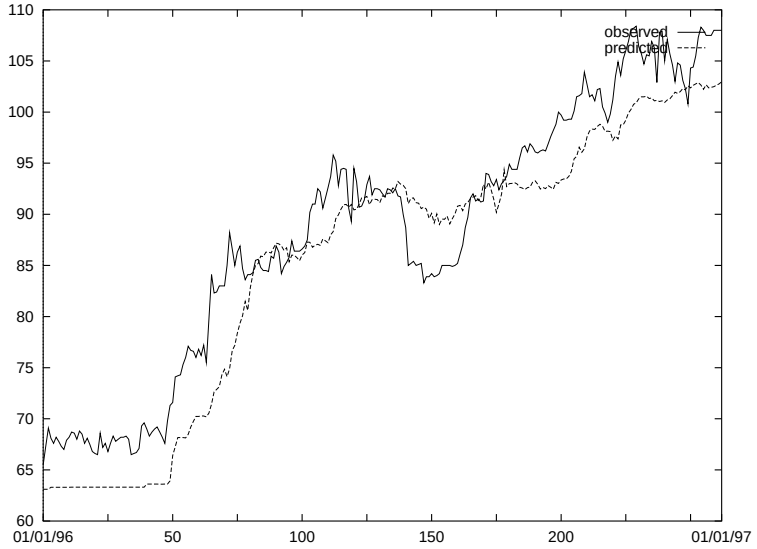

Figure 7.3: The results on data not seen before of the leader/follower technique trained on the MSE error and the actual stock.

7.2.2 Prediction of change in direction

The results that we obtained after training the network on the POCID can be seen in the next table. The first column indicates the network number. This number is arbitrary. The next columns give the MSE, normalized MSE, and the percentage of correct predicted directions for each network. These results were obtained by simulating one year's predictions. The stock values of Ahold of weekdays in the period of 01/01/96-01/01/97 are given to the network as inputs. We used a training set of 40 samples. After producing an output value,

the window is moved one day and the network is retrained on the updated training set.

network	MSE	normalized MSE	correct direction
01	31.02	0.0169	66.41%
02	33.36	0.0181	66.41%
03	31.62	0.0172	67.56%
04	36.47	0.0198	65.65%
05	38.45	0.0209	73.28%
06	36.32	0.0197	70.23%
07	36.64	0.0199	67.18%
08	36.93	0.0201	70.99%
09	31.79	0.0173	66.03%
10	34.82	0.0189	73.66%
average	34.74	0.0189	68.74%

As can be seen in figure 7.4, training on the POCID error pushes the values in the direction of the actual values. In the former diagram, the prediction cutted the actual diagram at several places, in stead of following it. Now the prediction follows the actual diagram in the same way as in the standard technique.

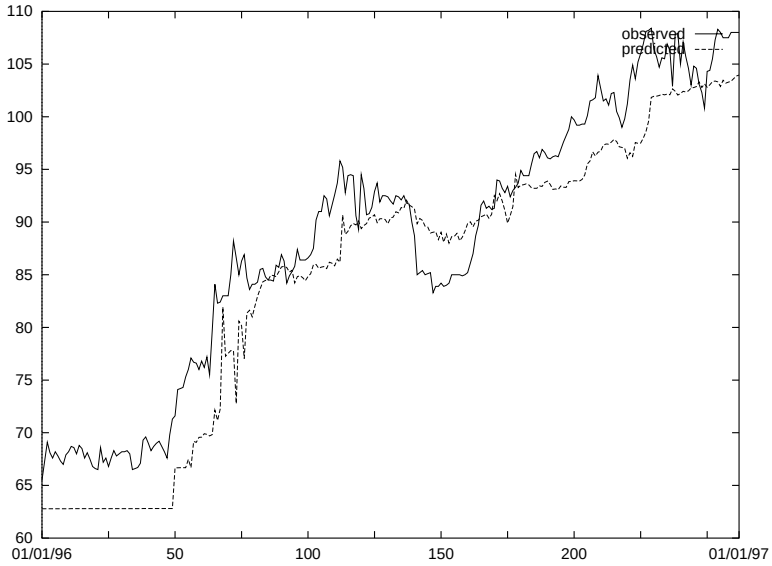

Figure 7.4: The results on data not seen before of the leader/follower technique trained on the POCID error and the actual stock.

7.2.3 Conclusion

The leader/follower technique using the MSE as error measure for training, produces better results if we consider the MSE. The leader/follower technique using the POCID error measure for training however, performs slightly better if we consider the percentage of correct predicted direction. Apparently training on the

POCID error (in combination with the MSE) does not increase the percentage of correct predicted directions when we use the leader/follower technique. Since the direction of the stock is more important than the exact value of the stock, training on the POCID is the best choice.

7.3 Error bars

It is possible to compute a confidence interval for each prediction. The idea is to train a number of networks, take the average prediction of these networks and then compute the confidence interval.

The purpose of the FFNs that we use for regression is to estimate the underlying mathematical function between input and output variables based on a finite number of data points possibly corrupted by noise. We have a data set of $p_{\texttt{data}}$ pairs $\{\vec{x}^\mu, t^\mu\}$ which are assumed to be generated according to

$$t(\vec{x}) = f(\vec{x}) + \xi(\vec{x}),$$

where $\xi(\vec{x})$ denotes noise with zero mean.
The output $o(\vec{x})$, given a new input vector $\vec{x}$, of a network that is trained on such a regression task can be interpreted as an estimate of the regression $f(\vec{x})$ (i.e. the mean of the target distribution given input $\vec{x}$). What we want to know for our prediction problem is the accuracy of the estimate of the true regression (i.e. the regression without noise).

When training a FFN on a particular sample, the weights are adjusted in order to minimize the error on the training set. Training is stopped when a certain error threshold had been reached. A set of n networks is trained and stopped on a certain data set. The output of network i on input vector $\vec{x}^\mu$ is written $o_i(\vec{x}^\mu) \equiv o_i^\mu$. As the estimate of the ensemble of networks for the regression $f(\vec{x})$ we take the average output

$$m(\vec{x}) \equiv \frac{1}{n}\sum_{i=1}^{n} o_i(\vec{x}).$$

Confidence intervals provide a way to quantify the confidence in the estimate $m(\vec{x})$ of the regression $f(\vec{x})$: consider the probability distribution $P(f(\vec{x}) \mid m(\vec{x}))$ that the true regression is $f(\vec{x})$ given the estimate $m(\vec{x})$.

Assume that the distribution $P(f(\vec{x}) \mid m(\vec{x}))$ is centered around $m(\vec{x})$. Neural networks are biased estimators. For example, neural networks trained on a finite number of examples will always have a tendency to oversmooth a sharp peak in the data. This can be seen in figure 7.2. This introduces a bias, which should be taken into account to get asymptotically correct confidence intervals. The hypothesis is that the bias component of the confidence intervals is negligible in comparison with the variance component.

First-order correct intervals can be derived by assuming a Gaussian distribution $P(f(\vec{x}) \mid m(\vec{x}))$. The variance of this distribution can be estimated from the variance in the outputs of the n networks:

$$\sigma^2(\vec{x}) \equiv \frac{1}{n-1}\sum_{i=1}^{n}[o_i(\vec{x}) - m(\vec{x})]^2.$$

Since the distribution is Gaussian, so is the inverse distribution $P(m(\vec{x}) \mid f(\vec{x}))$ to find the regression $m(\vec{x})$.

Following the procedure, the confidence intervals are:

$$m(\vec{x}) - c_{\texttt{confidence}}\sigma(\vec{x}) \leq f(\vec{x}) \leq m(\vec{x}) + c_{\texttt{confidence}}\sigma(\vec{x}),$$

where $c_{\texttt{confidence}}$ depends on the desired confidence level $1-\alpha$.
The confidence level that we use is 0.95, which results in 95% confidence intervals. The factors $c_{\texttt{confidence}}$ can be taken from a table with the percentage points of the Student's t-distribution with number of degrees of freedom equal to the number of networks that produced the prediction.

We used 50 networks to produce an estimate of the output. In figure 7.5 a diagram with errorbars is shown. Another possibility is to draw the confidence intervals using borders which show the limits of each confidence interval (see figure 7.6).

Remark

The confidence intervals only indicate the confidence of the *real* regression. The actual stock is the real regression *including noise*. Because of this, the actual stock sometimes exceeds the limits of the confidence intervals.

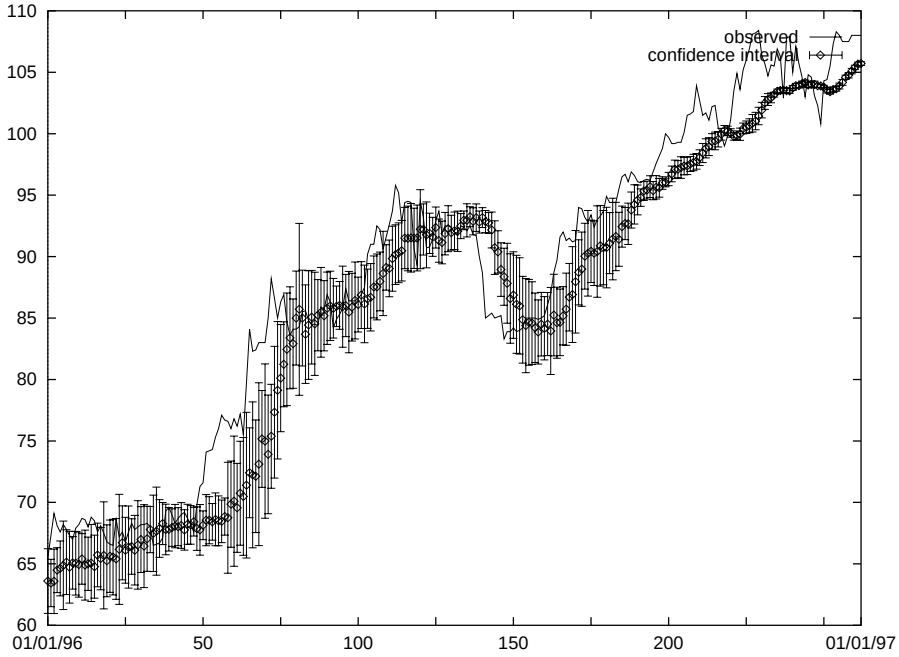

Figure 7.5: Diagram with errorbars to indicate the confidence interval

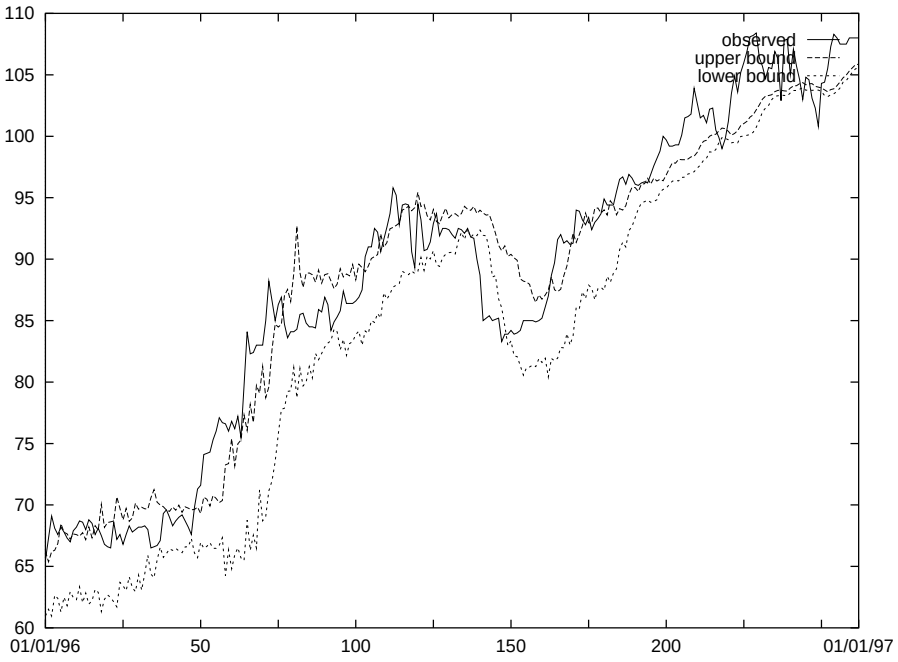

Figure 7.6: Diagram with borders to indicate the confidence interval

Chapter 8

Varying the size of the learning set

In this chapter the results of varying the size of the learning set will be discussed. The obtained results will be compared with the results of the naive prediction. The program builds a FFN, which produces an output based on 10 input values. We used the stock of Ahold for testing and producing results. The parameters of the FFNs are set for this specific stock. It is likely that for prediction of the stock of another company a new optimal parameter setting has to be found. This is caused by external factors, such as influence of other companies, oil trade, the frequency of trade in the stock of the company, etc. We will discuss the results of the two different approaches: standard technique and leader/follower technique.

8.1 Purpose of the analyses

On the data 263 outputs were produced. The network had to process a prediction for one weekday in 1996, based on the values of the 10 preceding weekdays. To make a prediction, the network is fitted on the preceding 40 weekdays. Training of the network is stopped, when the percentage of correct predicted directions decreases.

Each FFN produces the outputs for one year, namely each weekday from 01/01/96 - 01/01/97. For each technique 50 networks are trained. In the standard technique we have tried to find an optimum for the number of samples in the training set. The results are displayed in the following sections.

8.2 Naive prediction

An important issue is that the neural networks should outperform naive prediction. The naive prediction takes the last day as predicted value for the current day. If the naive prediction outperforms the FFNs, then it is useless to produce outputs by neural networks. The results that were obtained by using naive prediction are given in the next table. The columns give the MSE, normalized

MSE, and the percentage of correct predicted directions for each network. These results were obtained by taking one year's predictions (01/01/96-01/01/97).

MSE	normalized MSE	correct direction
1.52	0.000826	61.83%

8.2.1 Standard technique

The standard technique takes the 10 preceding weekdays from the stock of Ahold to predict an output. Then the time-window is moved one day so that the current day lies within the time-window, and the next day becomes the day to be predicted. The time-window is of fixed size: when the time-window is moved, then the first day of the time-window is deleted and a new day is added.

For each of the 263 days the MSE and the percentage of *predictions with the correct direction* were determined. A prediction of the value for weekday t has the correct direction if an increase or decrease was predicted with respect to the predicted value for weekday *t-1*, which is the same direction as the actual difference.

Sizes of time-windows

The FFNs trained on a data set of 40 samples turned out to perform best. This conclusion is based on the quality of the prediction in terms of MSE, POCID, and the time needed to predict all 263 values.

The FFNs trained on the complete data set produced outputs with smaller MSEs, but the execution time of the program increases considerably. Because of this increase of execution time, we consider this time-window size as unfit for prediction. In the stock market one has to react very quickly, and the FFNs that are trained with this time-window do not satisfy the requirements.

When we increased the number of samples in the data set, the FFNs turned out to decrease in percentage of correct predicted direction, and the execution time of the program increased (as can be expected). This can be seen in the following table. In this table the average MSE, POCID, and execution time for the networks trained on the different time-window sizes are given.

time-window	MSE	POCID	execution time
40	20.68	68.32%	4 min
50	27.69	65.27%	5 min
75	30.91	63.74%	11 min

The diagrams are shown in figures 8.1 to 8.3.

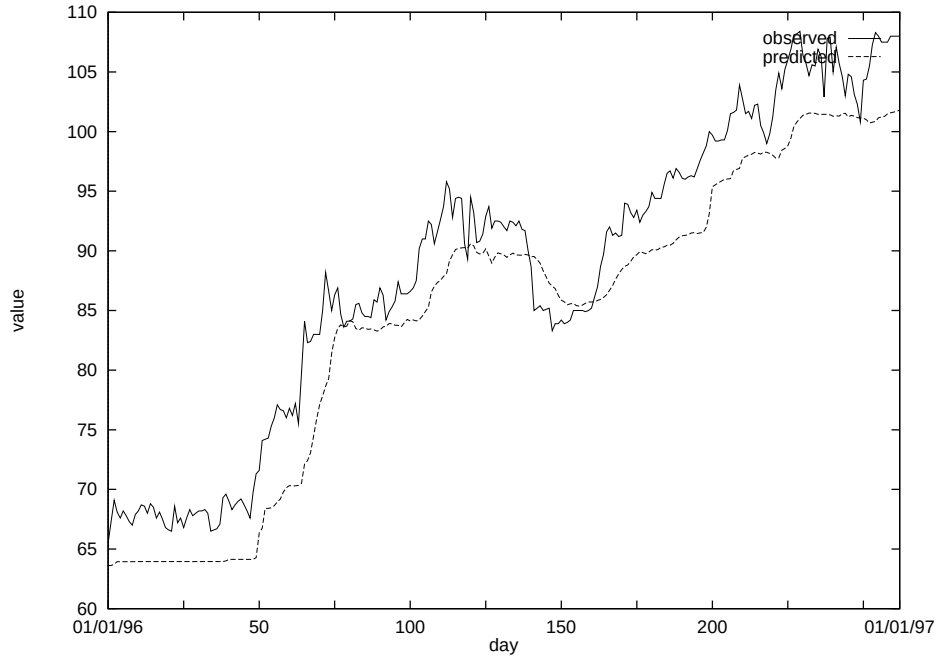

Figure 8.1: Diagram of networks trained on a data set of 40 samples, standard technique

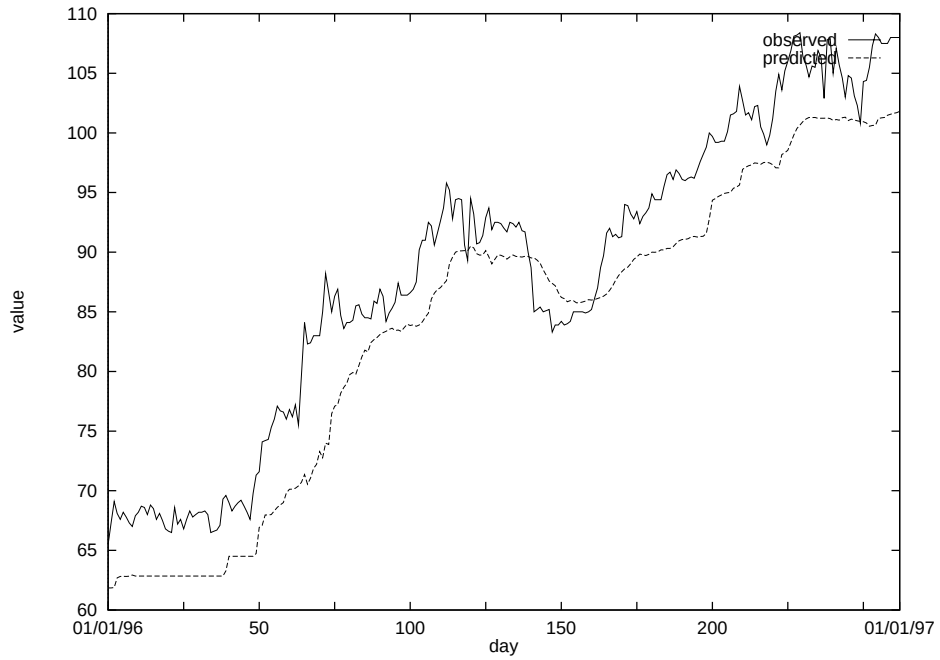

Figure 8.2: Diagram of networks trained on a data set of 50 samples, standard technique

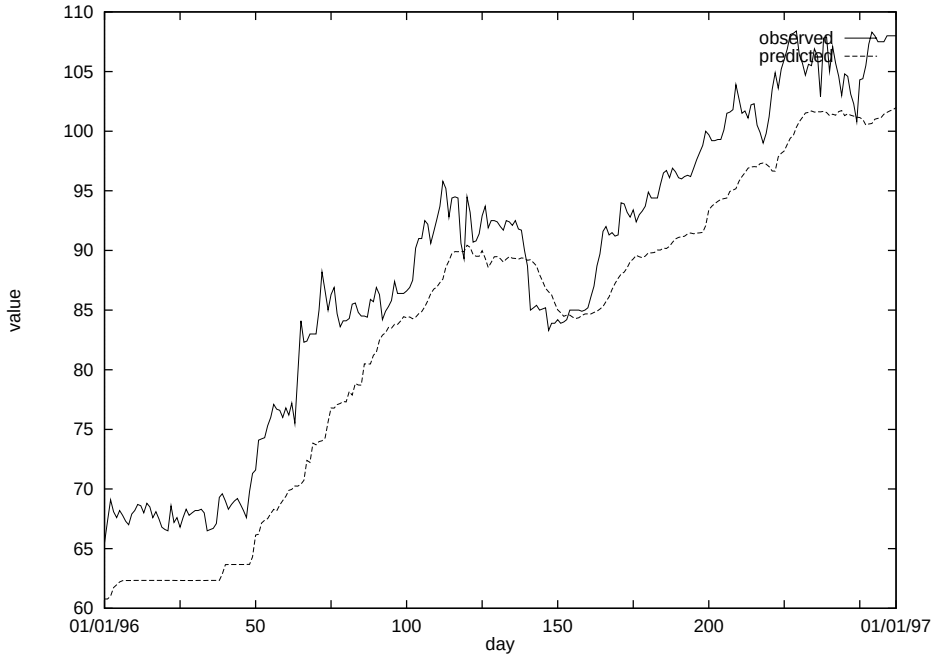

Figure 8.3: Diagram of networks trained on a data set of 75 samples, standard technique

8.2.2 Leader/follower technique

The leader/follower technique is based on the idea that the stock of a company can be influenced by the stock of other companies. The leader/follower technique takes the 10 preceding weekdays from the stock of the leaders of Ahold to predict an output for Ahold. As in the standard technique, the time-window is moved one day before processing the next output.

As discussed in chapter 7 the predicted stock of a network trained on the POCID as well as the MSE follows the actual stock better than the predicted stock of a network trained only on the MSE.

In the previous section FFNs trained on a time-window with 40 samples turned out to outperform networks trained on more samples. For these reasons we will only provide the results of the networks trained on the POCID and a time-window with 40 samples.

time-window	MSE	POCID	execution time
40	28.55	64.50%	6 min

The diagram is shown in figure 8.4.

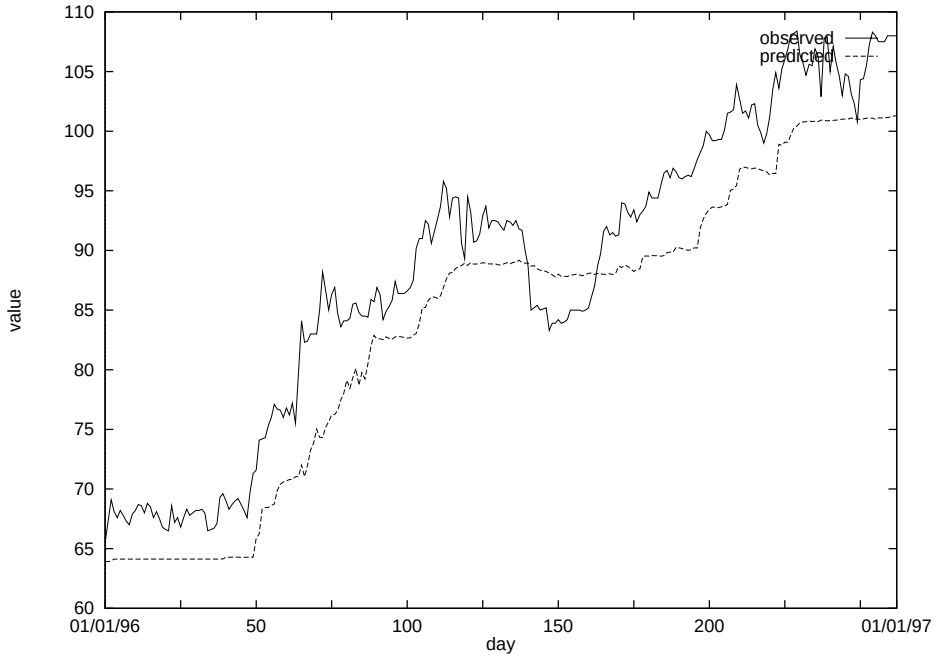

Figure 8.4: Diagram of networks trained on a data set of 40 samples, leader/follower technique

8.3 Conclusion

We have seen that the FFNs that use the standard technique perform much better than the FFNs that use the leader/follower technique if we consider both MSE and POCID. The cause is probably that the stock values of the leaders may deviate too much from the stock value of the follower. This causes the neural network to produce outputs that deviate from the stock values of the follower. There is only a slight difference in POCID.

However, the results that we obtain by introducing the leader/follower technique into the FFNs are worse than we expected. The standard technique obviously performs better. It is possible that better results can be obtained by using one of the other kinds of networks discussed in chapter 2.

Both techniques outperform the naive prediction, discussed in section 8.2, if we consider the POCID error. The techniques also outperform the experts (analysts) of the stock market, who can reach a percentage of approximately 56% on tops. Because the direction is much more important than the exact value, we can consider the FFNs to be better than the naive prediction.

Chapter 9

Ensembles of neural networks

In this chapter a comparison between different methods to combine predictions from neural networks will be given. One of these methods is balancing. This method is based on the analysis of the ensemble generalization error into an ambiguity term and a term incorporating generalization performances of individual networks. The method is described in [2].

A strategy to prevent a neural network from overfitting, is to stop training in an early stage of the learning process. The complete data set is split up into a training set and a validation set. Training is stopped when the error on the validation set starts increasing. The stability of the networks is highly dependent on the division in training and validation set, and also on the random initial weights and the chosen minimization procedure. This causes early stopped networks to be rather unstable: a small change in the data or different initial conditions can produce large changes in the prediction. Therefore, it is advisable to apply the same procedure several times starting from different initial weights. This technique is often referred to as training ensembles of neural networks.

Bagging

With bagging, the prediction on a newly arriving input vector is the average over all network predictions. Bagging completely disregards the performance of the individual networks on the data used for training and stopping.

Bumping

Bumping throws away all networks, except the one with the lowest error on the complete data set.

Balancing

Balancing is an intermediate form of bagging and bumping. Each network receives a weighting factor α_i which depends on the expected performance of the network on new values. This estimation is based on the performance of the network on the training and validation sets. The prediction of all networks on pattern ν is defined as the weighted average

$$\widetilde{m}^{\nu} \equiv \sum_{i=1}^{n_{\mathtt{run}}} \alpha_i \widetilde{o}_i^{\nu}$$

The goal is to find the weighting factors α_i, subject to the constraints

$$\sum_{i=1}^{n_{\mathtt{run}}} \alpha_i = 1 \;\; and \;\; \alpha_i \geq 0 \quad \forall i,$$

yielding the smallest possible generalization error

$$E_{test} \equiv \frac{1}{p_{\mathtt{test}}} \sum_{\nu=1}^{p_{\mathtt{test}}} (\widetilde{m}^{\nu} - \tilde{t}^{\nu})^2$$

We have to find reasonable estimates for these generalization errors based on the network performances on validation data. Once we have obtained these estimates, finding the optimal weighting factors α_i under the constraints is a straightforward quadratic programming problem.

9.1 Simulation

In this section a comparison of the methods for combining the neural network outputs will be given. The tests were run by M.H. Lamers. The networks used for combination of the results, are different from the ones used in chapter 8. The differences and the results are discussed in the following subsections.

9.1.1 Purpose of the analyses

On the data 263 separate neural network analyses were performed. Each of the analyses had to process the output for one weekday in 1996, based on the values of the preceding 10 days. To make a prediction, each model is fitted on 40 directly preceding weekdays.

For example: to process a prediction for 29/01/96 based on the 10 preceding weekdays, a model is fitted on the values of the 40 preceding weekdays (04/12/95 - 26/01/96). Consequently a prediction of the value on 29/01/96 is processed.

The network models

For each analysis 50 neural networks are trained on the appropriate 40 observations. Each network is a 3-layer network, with 10 input nodes, 4 nodes in the hidden layer, and 1 output node. The hidden nodes have a *tanh* transfer function, and the output node has a *linear* transfer function.

Training of the network was performed using the back-propagation algorithm, which was extended with the POCID learning rule. The POCID learning rule states that each step of the back-propagation algorithm can proceed normally if the direction of the prediction of the current observation was wrong. If, on the other hand, the direction was correct, then a weaker step of the algorithm will be done: the learning rate will then be multiplied by 0.1.

The back-propagation algorithm was stopped after 500 epochs of training, or if the MSE on the validation set increased for 4 subsequent steps. The learning rate and momentum parameters were 0.01 and 0.6 resp.

Normalization

To decrease the learning time of the neural networks the data for the analyses have been normalized. The complete data set has been transformed to its *Z-value*:

$$x = \frac{x - mean}{SD}$$

The resulting data set has an average of 0 and a standard deviation of 1.

Percentage of correct direction

For each of the 263 weekdays of 1996, three collective models were made from the 50 separate networks for bagging, bumping, and balancing methods. Note that for *each* of the 263 days three models were constructed. For each of these 263 predictions the MSE was determined, and the percentage of *predictions with the correct direction*.

A prediction of the value for weekday t has the correct direction if an increase or decrease was predicted with respect to the predicted value for weekday *t-1*, which is the same direction as the actual difference.

9.1.2 Differences

In this section the differences are discussed and the possible impacts:

transfer functions The FFNs used in this chapter have *tanh* transfer functions in the hidden layer and *linear* transfer functions in the output node. The FFNs used in chapter 8 contain only node with *sigmoid* transfer functions (including the output node). There is no difference in flexibility of the model caused by the use of *tanh* in stead of *sigmoid* functions. The network in this chapter may be slightly less flexible caused by the use of the linear output node. However, the difference is probably not perceptible.

learning rate and momentum Different values for learning rate and momentum may cause the FFNs to stop training in a different point of 'weight-space'. The impact of this difference is difficult to be estimated.

stop criterion The learning-algorithm of the networks in this chapter uses a validation stop criterion. The back-propagation algorithm is stopped when the MSE on the validation set increases in 3 subsequent steps. The validation data are selected from the training data using the 'bootstrapping' algorithm [2]. Furthermore, the back-propagation algorithm is proceeded for a maximum of 500 epochs. As a result of different stop criteria the networks may terminate in different configurations. The impact on the predictions is expected to be rather small.

normalization The output values for the FFNs in this chapter are normalized in the same way as the input values (Z-transformation), while in chapter 8 the output values are scaled to the domain [0:1]. This is necessary when a sigmoid transfer function in the output node is used. This difference has an effect on the computed MSE for both methods, but it has no impact on the percentages of predictions in the correct direction. It is possible to scale the MSEs back to a scale in which a comparison can be made.

learning In this chapter a new FFN is built for each of the 263 weekdays of 1996. In chapter 8 one network is trained for every subsequent weekday. For the next day the network is trained again, but the time-window has moved one day. This approach introduces continuity in the method, which is expected to be positive with respect to time series analysis. The impact of this difference is expected to be rather small, because the maximum number of epoch that the FFNs in this chapter are trained is rather large.

9.1.3 Results

The 50 FFN models that have been trained have to be combined to form a collective model. Using this model predictions can be made based on 10 preceding values. There are three methods to make this combination: *bagging, bumping, and balancing:*

individual The average individual generalization error, i.e. the generalization error we will get on average when we decide to perform only one run. It serves as a reference with which the other methods will be compared.

bagging The average of the predictions of 50 feed-forward networks is taken to produce an ensemble output.

bumping The best of 50 feed-forward networks is chosen, based on the best predictions made on the data in the modeling phase. This network is taken as the eventual model.

balancing By way of bootstrapping procedures an estimate of the performance of each of the networks on new data is made. Based on these estimates a weight is assigned to each network, using quadratic programming, which optimizes the weighted average of the estimated predictions. The ensemble output is the weighted average of the 50 networks.

The mean MSE of all 263 times 50 FFNs is 8.69. This value has been scaled back to the scale of the original data.

The results of the bagging, bumping, and balancing methods for combining the predictions of the 50 networks are expressed in the relative decrease of the MSE with respect to the MSE of the 50 separate predictions. Because this value has been computed for each of the 263 selected weekdays, the mean value and the standard deviation are reported.

Unfair bumping is a unfair method, where the actual value of the concerned weekday, to determine afterward which one of the 50 networks would have been the best choice. The results of this strategy are reported for comparison.

	Average decrease of the MSE (sd) (n = 263)
Bagging	28% (30%)
Bumping	39% (65%)
Balancing	37% (44%)
Unfair Bumping	88% (18%)

In the next tables the number and percentages of correct or wrong prediction of increase or decrease of the stock price are given. The rows indicate the actual directions, the columns indicate the according predicted directions. Subsequent observations which remained the same (no increase, no decrease) are given in a separate row.

bagging	decrease predicted	increase predicted
decrease	22 (8%)	79 (30%)
no change	5 (2%)	17 (6%)
increase	45 (17%)	94 (36%)

bumping	decrease predicted	increase predicted
decrease	36 (14%)	65 (25%)
no change	9 (3%)	13 (5%)
increase	47 (18%)	92 (35%)

balancing	decrease predicted	increase predicted
decrease	29 (11%)	72 (27%)
no change	10 (4%)	12 (5%)
increase	45 (17%)	94 (36%)

Next table shows the total number of correct and wrong predicted directions and the number of undefined changes in direction. If a direction is defined wrong when

```
(target(t) - target(t-1)) * (output(t) - output(t-1)) < 0
```

(cf. the "official" POCID definition), then all predictions where `target(t) - target(t-1) = 0` are considered 'correct'. The results of this approach are given in the last column.

	correct direction	wrong direction	undefined	POCID
bagging	116 (44%)	124 (47%)	22 (8%)	138 (53%)
bumping	128 (49%)	112 (43%)	22 (8%)	150 (57%)
balancing	123 (47%)	117 (45%)	22 (8%)	145 (55%)

Figures 9.1 to 9.3 show the outputs for each of the three methods. In each figure the actual stock is shown as well for comparison. The horizontal axis shows the 263 weekdays between 01/01/96 and 01/01/97, and the vertical axis shows the values that were predicted by each of the discussed methods.

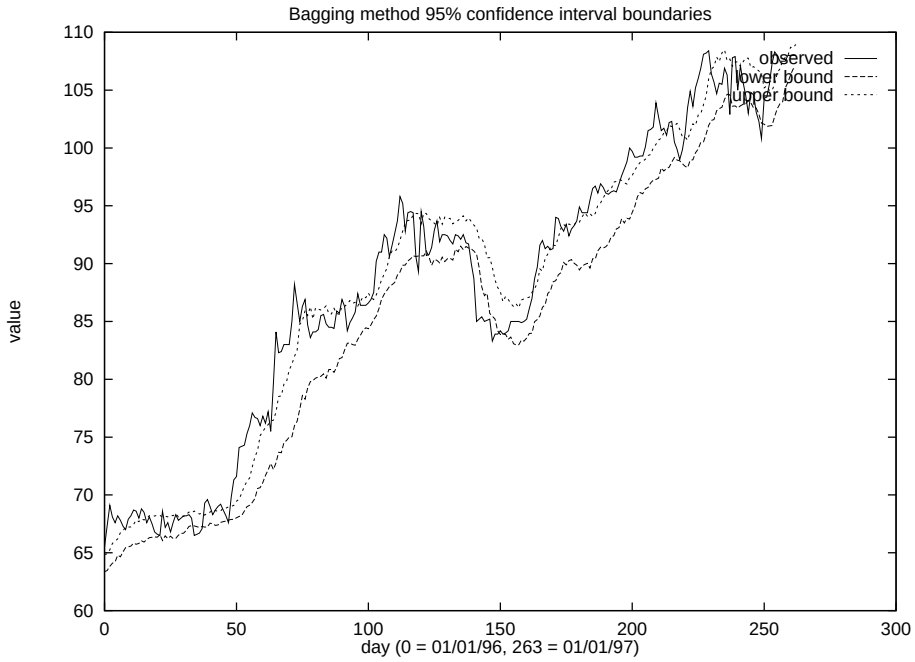

Figure 9.1: Output values for bagging

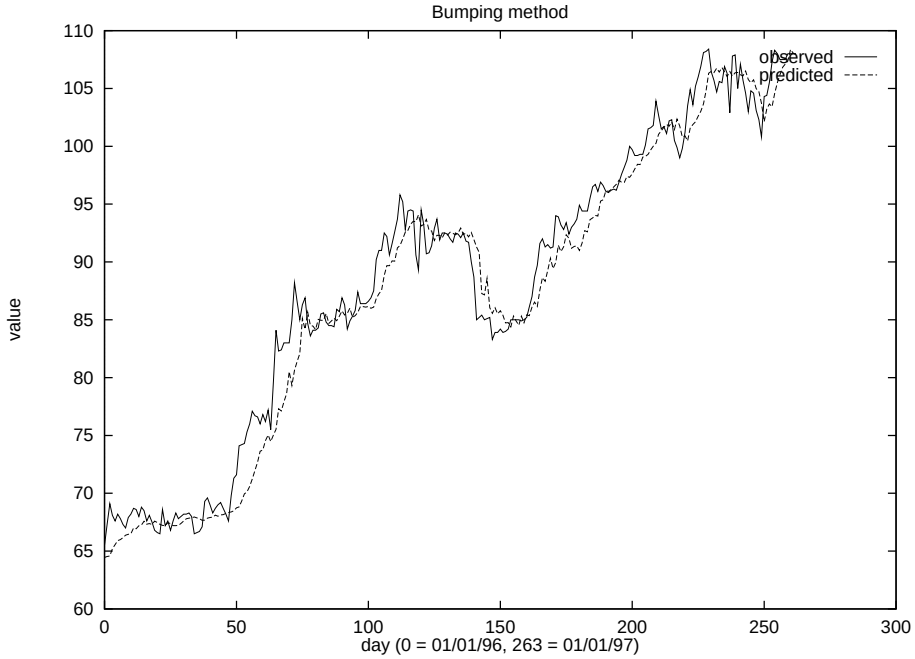

Figure 9.2: Output values for bumping

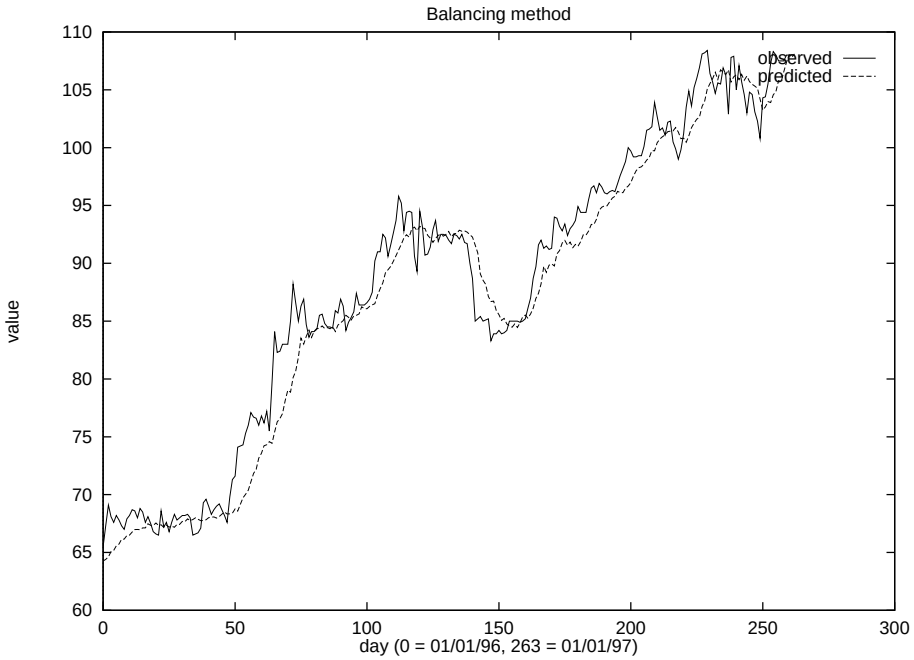

Figure 9.3: Output values for balancing

9.2 Conclusion

The results that we obtain by combining the FFNs perform much better than the separate FFNs if we consider the MSE. In subsection 9.1.3 we have seen that we can decrease the MSE considerably by combining the FFNs.

The ensemble of networks produces much worse outputs if we consider the POCID error compared to the results given in chapter 8, however. Since the direction of the stock is more important than the exact value, we consider the separate FFNs of chapter 8 to be better for our purposes.

Chapter 10

Future work

There are a number of things that can be improved:

- Speed of the program. The program can be speeded up by implementing a parallel version. Depending on the number of processors being used, the program can be speeded up considerably.
- Reading data from a database instead of reading them from file. This should make it easier to use data from different companies, for example when using the leader-follower technique. This could be done by embedded SQL.
- Implementing a graphical interface for the user, where the influence of changing some parameters will directly be depicted in some graphic or moving tube graph.
- Using the result of the stock price prediction in a *value-at-risk* model. This model expresses the chance at loss in one single value. This value should help investment companies in deciding whether to invest in an enterprise or not. This indication should be based on more factors of influence than only the stock price. For example, the overall economy of the country should be considered, as well as the previous profit or loss of the examined enterprise.
- For the problem of stock price prediction on a short term basis, the standard FFN seemed to perform good enough. This is the reason that we did not program or research the other options. If, however, the results of the prediction turn out to be unsatisfactory, some of the alternatives proposed in chapter 2 could be tried.

Appendix A

Documentation

A.1 Running the program

In this section directions for using the program will be given. The program can be called by typing `forecast`. The program name has to be followed by:

- a list of options
  for example, `forecast -c 2500 -e 0.1 -f -h 4 -i -m 0.001 -p -T -l AHOLD=1 -o out.val -t T_AHOLD=1`

or

- the name of a batch file which contains the options
  for example, `forecast batch`

These possibilities will be discussed in the following sections.
The program has to be called from the directory where all files with the data sets reside.

A.1.1 Options

The program can be run by calling `forecast` followed by a number of options, which have to meet with some conditions:

-a <alfa> The momentum term of back-propagation. The value must lie between -1.0 and 1.0 (otherwise an error will be generated).

-c <cycles> The maximum number of learning cycles that the network should perform. The number of cycles must be greater than zero.

-d <delay> The delay between the last input value and the prediction. The delay must be greater than zero.

-D <thrsh> The threshold to stop training. If the network produces outputs with an error below the threshold, then training can be stopped. The threshold should be greater than zero.

-e <eta> The learning rate of back-propagation. The value must lie between 0.0 and 1000 (otherwise an error will be generated).

-f Print the final performance of the network on the learning set and on the test set.

-F <fname> Perform only a prediction based on the values in fname. The network will not be trained or tested. It is required to enter a network filename too, because the prediction can only be done on a trained network.

-h <units> The number of hidden units of the network. The number of hidden units can influence the performance and learning speed of the network.

-i Print the initial performance of the network.

-I <units> The number of input units. This is the number of values that the network should take to predict some value in the future. If, however, the number of learning files is greater than one, the program will take the number of files as number of inputs. This is to enable the program to use the leader-follower technique.

-m <val> The mean squared error on which back-propagation will terminate. If no mean squared error is entered, the program will perform standard back-propagation and will terminate when the maximum number of cycles is reached.

-n <fname> The filename of some trained network. This can be used for prediction only, or it can be trained again.

-p Print the performance of the network after every ten back-propagation cycles.

-P Print all program variables with option.

-q Run the program silently (no messages during training).

-s <units> The number of hidden units in an optional second hidden layer. Normally, the network has one hidden layer, but a second hidden layer can be created.

-S <sample> Number of samples to train the network, i.e. the size of the learning set.

-T Print the CPU-time that the program used.

-u <min> The lower bound for random generator of weights.

-U <max> The upper bound for random generator of weights. The initial values of the weights will lie between <min> and <max>.

-y Run a simulation of the last year.

The following options are mandatory:

-l <fname>=d One filename followed by a delay, or a list of filenames followed by their corresponding delays and separated by comma's. These are the files on which the network will be trained (the learn files).

-o <fname> The filename of the file to which the predicted values should be written.

-t <fname>=d One filename followed by a delay, or a list of filenames followed by their corresponding delays and separated by comma's. These are the files on which the network will be tested. The number of test files must be equal to the number of learn files. They must be presented to the program in the same order and must have the same delay.

A.1.2 Batch file

Another possibility for running the program is to call `forecast` followed by a batch file. This file contains all options discussed in subsection A.1.1. An example batch file:

```
hunits = 4
iunits = 10
cycles = 10000
delay = 1
thrsh = 0.075
options = 302
eta = 0.5
alfa = 0.9
sunits = 0
samples = 75
minunif = -0.01
maxunif = 0.01
final_mse = 0.0001
learndata =  AHOLD=1
outtest = out.val
netfname = NULL
testdata = T_AHOLD=1
```

We will now give a list of correspondences between the used option names and the options given in subsection A.1.1.

`hunits`	-h <units>
`iunits`	-I <units>
`cycles`	-c <cycles>
`delay`	-d <delay>
`thrsh`	-D <thrsh>
`options`	one or more of of -f, -i, -p, -P, -q, -T, -y this results in a number (see next table)

`eta`	-e <eta>
`alfa`	-a <alfa>
`sunits`	-s <units>
`samples`	-S <sample>
`minunif`	-u <min>
`maxunif`	-U <max>
`final_mse`	-m <val>
`learndata`	-l <fname>=d
`outtest`	-o <fname>
`netfname`	-n <fname>
`testdata`	-t <fname>=d

The 'options' field must contain a value. Next table provides the possible values for the 'options' field, together with their meanings. To get a combination of these options, one has to add the values for the desired options.

value	meaning
2	Print the performance of the network after every ten back-propagation cycles. (-p)
4	Print the initial performance of the network. (-i)
8	Print the final performance of the network on the learning set and on the test set. (-f)
32	Print the CPU-time that the program used. (-T)
128	Run the program silently (no messages during training). (-q)
256	Print all program variables with option. (-P)
512	Run a simulation of the last year. (-y)

Standard technique

For the standard technique we need only one file for the learning set and one file for the testing set. The filenames of the data sets have to be followed by '=' and a delay. The delay is the number of days between each input value in the samples. For example, if the delay is one, then subsequent days will be used for constructing the input vector. It is important that the delay of the testing set equals the delay of the learning set.

An example of a batch file which contains the options for the standard technique:

```
hunits = 4
iunits = 10
cycles = 10000
delay = 1
thrsh = 0.075
options = 302
eta = 0.5
alfa = 0.9
sunits = 0
```

```
samples = 75
minunif = -0.01
maxunif = 0.01
final_mse = 0.0001
learndata =  AHOLD=1
outtest = out.val
netfname = NULL
testdata = T_AHOLD=1
```

Leader/follower technique

For the leader/follower technique we need multiple files for the learning set and multiple file for the testing set. If the leader/follower technique is used, then the first file of the list is considered to be the one that has to be predicted. The filenames of the data sets have to be followed by '=' and a delay. The delay that is indicated is the found number of days between an increase/decrease of a leader and the reaction of the follower. It is important that the delays of the testing sets equal the delays of the learning sets for each file.

An example of a batch file which contains the options for the leader/follower technique:

```
hunits = 4
iunits = 10
cycles = 10000
delay = 1
thrsh = 0.075
options = 302
eta = 1.0
alfa = 0.9
sunits = 0
samples = 40
minunif = -0.01
maxunif = 0.01
final_mse = 0.0001
learndata = AHOLD=1,LEADER1=3,LEADER2=5,LEADER3=1,LEADER4=5,
LEADER5=1,LEADER6=3,LEADER7=2,LEADER8=5
outtest = out.val
netfname = NULL
testdata = T_AHOLD=1,T_LEADER1=3,T_LEADER2=5,T_LEADER3=1,
T_LEADER4=3,T_LEADER5=1,T_LEADER6=3,T_LEADER7=2,T_LEADER8=5
```

A.2 Data sets

The program has to be called from the directory where all files, which contain the data sets, reside. Each line of the data set must contain a date between quotation marks, followed by a value or `NA` (Not Available). The date and the

value are separated by a comma, and only one day per line is allowed. The date has the following format: "dd/mm/yy" or "dd-mm-yy". That is, two positions for the day, two positions for the month, and two positions for the year. The value may contain a point, but never a comma. The program demands that all lines contain subsequent dates, except for the weekends which may be skipped. This means that each line has the following format:

`"dd/mm/yy"`	this day's value	The date and value of the index
	OR	
`"dd-mm-yy"`	this day's value	The date and value of the index

The data set may be preceded by any form of comment, provided that the second position of the line is not a number.

The data sets can be retrieved from the DataStream provider. ING ITResearch has a dish antenna which provides real-time data, that for security reasons, are presented for test purposes with a delay of 15 minutes. The format of the data sets which are provided by DataStream is appropriate for our program. The data sets provided by DataStream have the following format:

`"Name"`	name of enterprise	For recognizing the company
`"Code"`	code for enterprise	The code for retrieving the data
`"Currency"`	currency	The currency of the values
`"dd/mm/yy"`	this day's value	The date and value of the index

The files which are provided by DataStream have the following format:

```
"Name","ING CERTS."
"Code",531865
"Currency","FL"
"28/02/91",NA
"01/03/91",NA
"04/03/91",46.75
"05/03/91",47.84
"06/03/91",48.73
"07/03/91",48.24
"08/03/91",48.04
"11/03/91",46.95
"12/03/91",46.75
"13/03/91",46.46
"14/03/91",47.44
"15/03/91",47.84
```

This format is suitable for the program. The first three lines will then be skipped by the program. Or the user could delete the first three lines.

Bibliography

[1] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[2] T. Heskes. Balancing between bagging and bumping. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, 1996. MIT Press.

[3] T. Heskes. Practical confidence and prediction intervals. In M. Jordan M.Mozer and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, 1996. MIT Press.

[4] A. Hoekstra, M.A. Kraaijveld, D. de Ridder, and W.F. Schmidt. *The Complete SPRLIB & ANNLIB.* Delft University of Technology, The Netherlands, 199.

[5] A.G Ivakhnenko. Polynomial theory of complex systems. In *IEEE Transactions on Systems, Man and Cybernetics; SMC - 1(4): 364-378.* October 1971.

[6] C. Jongeneel. Neurale netwerken verslaan het brein van de beursanalist. *De Volkskrant*, 2 november 1996.

[7] M. Kendall and J.K. Ord. *Time series.* Edward Arnold, 1990.

[8] M.H. Lamers. Multiscale image segmentation with feedforward neural networks. Master's thesis, Utrecht University, October 1993.

[9] G. Oosterom. Time series prediction using simple recurrent networks. Master's thesis, Utrecht University, August 1994.

[10] W.H. Press, S.A. Teukolsky, W.T. Vetterling, , and B.P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge University Press, 1992.

[11] V. Rao and H. Rao. *C++ Neural Networks and Fuzzy Logic.* MIS:Press, 1995.

[12] J. van der Vlugt. The group method of stock price prediction. Master's thesis, Delft Technical University, December 1994.

[13] P.D. Wasserman. *Advanced Methods in Neural Computing.* Van Nostrand Reinhold, 1993.

[14] P.R. Water, E.J.H. Kerckhoffs, and H. Koppelaar. Financial models and parallel algorithms, impact progress report no. 1. Technical report, Delft Technical University, 1997.