

# Stock Prediction – A Neural Network Approach

Karl Nygren  
karlnyg@kth.se

28th March 2004

**Master Thesis**  
**Royal Institute of Technology, KTH**  
**Supervisor: Prof. Kenneth Holmström**  
**Examiner: Dr. Torkel Erhardsson**

## **Abstract**

Predicting stock data with traditional time series analysis has proven to be difficult. An artificial neural network may be more suitable for the task. Primarily because no assumption about a suitable mathematical model has to be made prior to forecasting. Furthermore, a neural network has the ability to extract useful information from large sets of data, which often is required for a satisfying description of a financial time series.

This thesis begins with a review of the theoretical background of neural networks. Subsequently an Error Correction Neural Network (ECNN) is defined and implemented for an empirical study. Technical as well as fundamental data are used as input to the network. One-step returns of the Swedish stock index and two major stocks of the Swedish stock exchange are predicted using two separate network structures. Daily predictions are performed on a standard ECNN whereas an extension of the ECNN is used for weekly predictions.

In benchmark comparisons, the index prediction proves to be successful. The results on the stocks are less convincing, nevertheless the network outperforms the naive strategy.

## Sammanfattning

Att prediktera börsdata med traditionell tidsserieanalys har visat sig vara svårt. Ett artificiellt neuralt nätverk kan vara mer passande för uppgiften. Främst därför att inga antaganden om en passande matematisk modell måste göras innan prediktering. Vidare har ett neuralt nätverk förmågan att extrahera användbar information från stora datamängder, vilket ofta är nödvändigt för en tillfredsställande beskrivning av en finansiell tidsserie.

Det här examensarbetet börjar med en genomgång av teorin bakom neurala nätverk. Därefter definieras och implementeras ett felkorrigerande neuralt nätverk (ECNN) för en empirisk studie. Både tekniska- och fundamentala data används som indata till nätverket. Enstegsavkastningar för Generalindex samt två stora aktier på Stockholmsbörsen predikteras med två separata nätverksstrukturer. Dagliga prediktioner utförs på en standard ECNN medan en utökad variant av ECNN används för veckoprediktioner.

Vid jämförelser med andra strategier visar sig prediktionen av index vara framgångsrik. Resultaten för aktierna är mindre övertygande, likväl presterar nätverket bättre än den naiva strategin.

### **Acknowledgements**

First and foremost I thank Prof. Kenneth Holmström at Tomlab Optimization AB for initialising this project. Tomlab Optimization AB has been supportive with financial data, computer and office facilities. Furthermore I am grateful to Dr. Hans-Georg Zimmermann at Siemens AG Corporate Technology Department for giving me an excellent introduction to the SENN software. I also acknowledge staff at Siemens AG for technical support. Finally, I am thankful to Dr. Thomas Hellström for useful comments.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Neural Networks</b>	<b>5</b>
2.1	The Single Neuron . . . . .	5
2.1.1	Activation Functions . . . . .	7
2.2	Network Structures . . . . .	8
2.2.1	Feed-forward Networks . . . . .	8
2.2.2	Recurrent Networks . . . . .	9
2.3	Brief History . . . . .	9
2.4	Traditional Time Series Analysis . . . . .	11
2.5	Benefits with Neural Networks . . . . .	11
2.6	Applications Outside Finance . . . . .	12
2.7	Neural Networks in Finance . . . . .	13
2.7.1	Motivation . . . . .	13
2.7.2	Data . . . . .	13
<b>3</b>	<b>Error Correction Neural Networks</b>	<b>14</b>
3.1	Mathematical Description . . . . .	14
3.1.1	Variants and Invariants . . . . .	16
3.2	Training . . . . .	17
3.2.1	Backpropagation . . . . .	17
3.2.2	Learning Algorithm . . . . .	17
3.2.3	Cleaning . . . . .	19
3.2.4	Stopping Criteria . . . . .	19
3.2.5	Error Function . . . . .	20
<b>4</b>	<b>Evaluation</b>	<b>21</b>
4.1	Benchmarks . . . . .	21
4.2	Performance Measures . . . . .	22
<b>5</b>	<b>Empirical Study</b>	<b>23</b>
5.1	Data Series . . . . .	23
5.2	Technical Considerations . . . . .	24
5.3	Training Procedure . . . . .	25

5.4	Implementation . . . . .	25
<b>6</b>	<b>Results</b>	<b>26</b>
6.1	Daily Predictions . . . . .	26
6.2	Weekly Predictions . . . . .	29
6.3	Comparison to Benchmarks . . . . .	31
<b>7</b>	<b>Discussion</b>	<b>33</b>
7.1	Possible Improvements . . . . .	33
<b>A</b>	<b>The Error Backpropagation Algorithm</b>	<b>37</b>
<b>B</b>	<b>Preprocessing</b>	<b>40</b>

# Chapter 1

## Introduction

Predicting the stock market is not a simple task. Mainly as a consequence of the close to random-walk behaviour of a stock time series. Different techniques are being used in the trading community for prediction tasks. In recent years the concept of *neural networks* has emerged as one of them.

A neural network is able to work parallel with input variables and consequently handle large sets of data swiftly. The principal strength with the network is its ability to find patterns and irregularities as well as detecting multi-dimensional non-linear connections in data. The latter quality is extremely useful for modelling dynamical systems, e.g. the stock market. Apart from that, neural networks are frequently used for pattern recognition tasks and non-linear regression.

This thesis gives an introduction to the theory of neural networks and a corresponding mathematical description. An Error Correction Neural Network (ECNN) is built and implemented for an empirical study. Standard benchmarks are used to evaluate the network's ability to make forecasts.

The objective of this study is to conclude whether an ECNN could be successfully used as decision support in a real trading situation. The matters of buy-sell signals, transaction costs and other trading issues are *not* considered.

## Chapter 2

# Neural Networks

The foundation of neural networks in a scientific sense begins with biology. The human brain consists of an estimated 10 billion *neurons* (nerve cells) and 6000 times as many *synapses* (connections) between them [Hay94]. All information taken in by a human is processed and assessed in this particular part of the body. A neuron in itself is relatively slow compared to a silicon logic gate. However, this amazing amount of neurons and synapses suites as compensation. Thus the brain operates as nothing less than a complex, non-linear and parallel computer [Gro02]. With this notion present we are ready to describe a neural network mathematically.

### 2.1 The Single Neuron

Let us begin with a fundamental description of the human brain. A process begins when stimulus is received from the environment. The *receptors* transform this information to electrical impulses and transmit them to the neural network (neurons and synapses) (Fig. 2.1). After evaluation inside the network, actions are decided and impulses are being sent out to the *effectors*.

Both biological and artificial neurons are elementary information processing units. Therefore also fundamental building blocks of a neural network [Gro02]. The artificial neuron is best illustrated by analogy with the biological neuron. Fig. 2.2 depicts the artificial neuron. We see that the connections (synapses)  $w_i$  transfer the signals (stimulus)  $u_i$  into the neuron.  $w_i$  can be interpreted as a weight representing the “importance” of that specific input  $u_i$ . Inside the neuron the sum of the weighted inputs  $w_i u_i$  is taken. Given that this sum  $u$  is greater than an externally applied threshold  $\theta$ , the neuron emits an output  $z$ .  $z$  is either continuous or binary valued, depending on the *activation function* (or *squashing function*). In most cases one choose an activation function that limits the range of the neuron’s output to the interval  $[0, 1]$  or  $[-1, 1]$ .

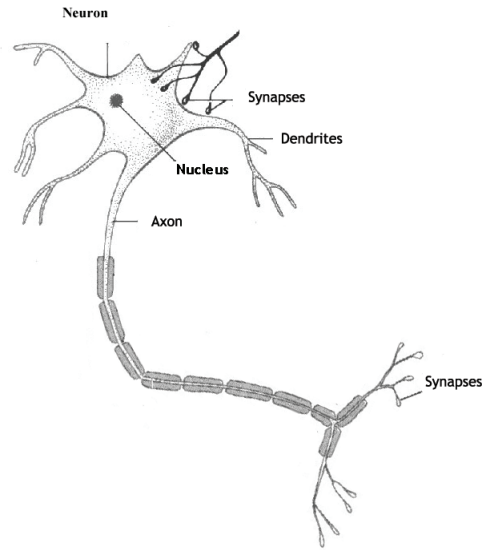


Figure 2.1: The biological neuron.

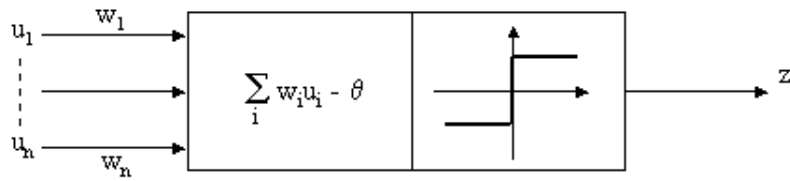


Figure 2.2: The artificial neuron with a threshold function.

In mathematical terms the following equations gives a dense description of the neuron:

$$y = \sum_{i=1}^n w_i u_i - \theta \quad (2.1)$$

and

$$z = \psi(y) \quad (2.2)$$

where  $y$  is the net input and  $\psi(\cdot)$  the activation function.

### 2.1.1 Activation Functions

In neural computing almost exclusively three different types of activation functions are being used:

(i) *the threshold function*

$$\psi(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0, \end{cases} \quad (2.3)$$

(ii) *the piecewise linear function*

$$\psi(x) = \begin{cases} 1 & x \geq \frac{1}{2} \\ x & -\frac{1}{2} < x < \frac{1}{2} \\ 0 & x \leq -\frac{1}{2} \end{cases} \quad (2.4)$$

and

(iii) *sigmoid functions.*

An example of a sigmoid is the *logistic function*

$$\psi(x) = \frac{1}{1 + e^{-a(x)}} \quad (2.5)$$

where  $a$  controls the slope.

Eq. 2.3 describes the “true-or-false” property and is often referred to as the McCulloch-Pitts model (see Sec. 2.3). The piecewise linear function is similar to the threshold function with an additional linear region. The most popular activation function though is the sigmoid which shows both a linear and non-linear behaviour. This function is continuous and differentiable which is worth noticing. Fig. 2.3 shows all three activation functions mentioned above.

In the empirical study (Sec. 5) we used another sigmoid, the *hyperbolic tangent function*.

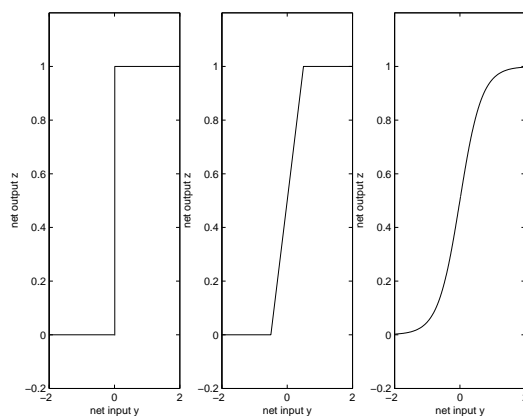


Figure 2.3: From left to right: the threshold function, the piecewise linear function and the logistic function ( $a=3$ ).

## 2.2 Network Structures

The importance of the network design (arrangement between neurons and synapses) is not to be underestimated. There is a tight relationship between the learning algorithm and network structure which makes the design central [Hay94, p. 18].

Two different types of neural networks can be distinguished, *feed-forward* and *recurrent* networks. The ECNN, described in Section 3, may be viewed as a recurrent architecture.

### 2.2.1 Feed-forward Networks

A typical neural network consists of layers. In a *single layered* network there is an input layer of source nodes and an output layer of neurons. A *multi-layer* network has in addition one or more hidden layers of hidden neurons. Both types of networks are displayed in Fig. 2.4. Extra hidden neurons raise the network's ability to extract higher-order statistics from (input) data. This is a crucial quality, especially if there is a large input layer.

Furthermore a network is said to be *fully connected* if every node in each layer of the network is connected to every other node in the adjacent forward layer. In a *partially connected* structure at least one synaptic connection is missing.

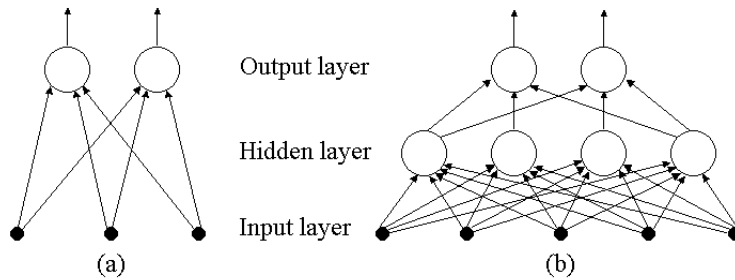


Figure 2.4: A feed-forward network with a single output layer of neurons (a) and a fully connected feed-forward network with one hidden layer and one output layer (b).

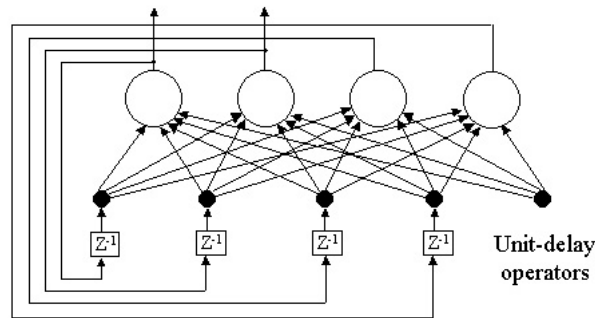


Figure 2.5: A recurrent network with hidden neurons.

### 2.2.2 Recurrent Networks

As the name suggests, a recurrent network supplies feedback to the network, i.e. at least one feedback loop exists. Fig. 2.5 offers a visualisation.

## 2.3 Brief History

A complete and satisfying review of historical developments in the research area of neural networks is beyond the scope of this thesis. Instead we have focused on a few important breakthroughs throughout history. For a thorough survey [Hay94] is recommended.

McCulloch and Pitts work on neural networks, published in 1943, still is a cornerstone in the theory of neural networks. They made an attempt to understand and describe the brain functions by mathematical means. McCulloch and Pitts

used their neural networks to model logical operators. Contemporary developments in the field of computer science were closely related.

In 1949 Hebb proposed that the synaptic connections inside the brain are constantly changing as a person gains experience. In other words, synapses are either strengthened or weakened depending on whether neurons on either side of the synapse are activated simultaneously or not. Among psychologists Hebb made an instant impact but network modellers have generally shown little interest in his work.

In the late fifties Rosenblatt introduced the concept of the *perceptron*. Basically, the perceptron, which works as a pattern-classifier, is a more sophisticated model of the neuron developed by McCulloch and Pitts. Depending on the amount of neurons incorporated, the perceptron can solve classification problems with various number of classes. For a correct classification the classes have to be linearly separable which is a major setback. This was shown by Minsky and Papert in 1969. Minsky and Papert also raised the issue of the *credit-assignment* problem (see Sec. 2.5) related to the multi-layer perceptron.

During the next decade the general interest in neural networks dampened, mainly as a direct consequence of the results reported in the late sixties. Certainly the lack of powerful experimental equipment (computers, work stations etc.) also had an influence on the decline.

The interest in neural networks was to be renewed though. In 1982 Kohonen introduced the *Self-Organising Map* (SOM). SOM:s use an unsupervised learning algorithm for applications in specifically data mining, image processing and visualisation. As a basic description one can say that high-dimensional data is transformed and organised in a low-dimensional output space. The same year Hopfield built a bridge between neural computing and physics. A *Hopfield* network (consists of symmetric synaptic connections and multiple feedback loops) which is initialised with random weights eventually reaches a final state of stability. From a physicists point of view a Hopfield network corresponds to a dynamical system falling into a state of minimal energy.

Two years later the *Boltzmann machine* was invented. As the name suggests, the work of Ludwig Boltzmann in thermodynamics was a source of inspiration. This neural network utilises a stochastic learning algorithm based on properties of the Boltzmann distribution.

The discovery of the *backpropagation algorithm* (see Sec. 3.2.1) in 1986 proved crucial for the revival of neural networks. Rummelhart, Hinton and Williams got the credit but it showed that Werbos already in 1974 had introduced the error backpropagation in his PhD thesis. This learning algorithm is unchallenged as the most influential learning algorithm for training of multi-layer perceptrons.

We conclude this section with the *Radial-Basis Function* (RBF) network, which was brought forward by Broomhead and Lowe in 1988. The RBF network emerged as an alternative to the multi-layer perceptron in the search of a solution to the multivariate interpolation problem. By using a set of symmetric non-linear functions<sup>1</sup> in the hidden units of a neural network new properties could be explored. Work by Moody and Darken (presented in 1989) on how to estimate parameters in the basis functions has contributed significantly to the theory.

## 2.4 Traditional Time Series Analysis

In conventional time series analysis instructions and rules are central. A mathematical formula defines the dynamics. One pick a model that is assumed to be applicable for the present task, e.g. the well known Auto Regressive Moving Average (ARMA) model [BD02].

Contrarily neural networks do not perform according to preset rules. When displayed to data the network gains experience, learns from regularities in the past and sets its own rules. Data are not described explicitly in mathematical terms. Neural networks are unique in that sense.

## 2.5 Benefits with Neural Networks

Neural networks have several advantages. Most important is the ability to learn from data and thus potential to generalise, i.e. produce an acceptable output for previously unseen input data (important in prediction tasks). This even holds (to a certain extent) when input series contain low-quality or missing data. Another valuable quality is the non-linear nature of a neural network. Potentially a vast amount of problems may be solved (see Sec. 2.6). Furthermore no expert system (typically a programmer coding rules in a computer program) is needed which makes the network extremely flexible to changes in the environment. One only has to retrain the system.

Regarding downsides, the *black-box-property* first springs to mind. Relating one single outcome of a network to a specific internal decision (known as the credit-assignment problem [Gro02, Hay94]) is very difficult. Noisy data also reinforce the negative implications of establishing incorrect causalities, *overtraining* (or *overfitting*), which will harm generalisation. Finally, a certain degree of knowledge in current subject is required as it is not trivial to assess the relevance of chosen input series.

---

<sup>1</sup>A frequently used function is the *Gaussian bump*.

A short summary of benefits and drawbacks:

- + generalisation ability and robustness
- + mapping of input/output
- + no assumptions of model has to be made
- + flexibility
- black-box property
- overfitting
- expertise for choice of input
- training takes a lot of time

## 2.6 Applications Outside Finance

Neural networks are being used in widespread areas. Our intention is to give a few examples of how neural networks can be used in practice.

The Newton Message Pad (a portable pen-based computer) introduced by Apple in the mid 90's implemented an artificial neural network for *character recognition*. The message pad is widely regarded as the world's first truly useable handwriting recognition system. Black and white pixels representing handwritten characters are transformed to its corresponding digital character. A large multi-layer perceptron is able to accomplish this task [Hay94]. See [PG02] for a practical application of character recognition. Related techniques are *speech recognition* and *speech production*. The former one was successfully developed in the NETtalk-system [Fau94]. The system converts English texts into phonetic and stress data, eventually used to produce synthetic speech.

In medicine neural networks are used to diagnose heart failures. For this complex disease a neural network's potential to detect multi-dimensional relationships in EKG data is of great advantage [Ati00].

*Signal processing* is another interesting area for neural networks. Very early nets were used to reduce noise on telephone lines by applying an adaptive filter at the end of a long-distance line [Fau94].

The last example is taken from the world of physics. An *Optimal Linear Associative Memory* (OLAM) neural network can be used to identify radioisotopes from their gamma-ray spectra. The OLAM determines the composition of a sample when the unknown spectrum is a linear superposition of known spectra. One feature of this technique is that the whole spectrum is used in the identification process instead of individual peaks only [KK94].

## 2.7 Neural Networks in Finance

Neural networks can be applied to all sorts of financial problems, not only stock prediction tasks. Forecasts of yield curves<sup>2</sup>, exchange rates, bond rates etc. are common. See [ZGN00] for a report of a successful prediction of the semi-annual development of the German yield curve.

Issues highlighted in Section 2.7.1 and 2.7.2 applies to stock prediction. Bare in mind other financial problems show similar characteristics.

### 2.7.1 Motivation

The principal motivation for the neural network approach in stock prediction is twofold:

- stock data is highly complex and hard to model, therefore a non-linear model is beneficial
- a large set of interacting input series is often required to explain a specific stock, which suites neural networks

It is also possible to approach the prediction task from the angle of economics. Grothmann (2002) [Gro02, p. 46] suggests the following viewpoint: Each single neuron represents a market participant's decision process. Hence a neural network represents interacting decisions among all participants in the market. Thus a neural network is a complete description of the financial market in itself.

This viewpoint gives an attractive mixture of the mathematical theory of neural networks and economics.

### 2.7.2 Data

Stock forecasting have some central aspects. Almost exclusively one wishes to predict returns rather than actual stock prices. Primarily, because the evaluation phase (see Sec. 4) becomes easier. Another reason is that it facilitates stabilisation of the model over a long period of time [Hel98, p. 19]. In practice, data are transformed prior to modelling (see Sec. 5.2).

Regarding actual data, the sometimes poor quality is well documented. Frequently one has to handle missing data points or even discontinuous time series. A convenient way to work around these difficulties is by letting the network accept missing data. Many times a significant part of the underlying dynamics can be learned anyway.

---

<sup>2</sup>A yield curve shows the relationship between yields and maturity dates for a set of similar bonds at a given point in time.

## Chapter 3

# Error Correction Neural Networks

Most dynamical systems contain both an autonomous part and a part governed by external forces. Many times relevant external forces may be hard to identify or data could be noisy. As a consequence a correct description of the dynamics may be impossible to get. A remedy for a better model description is to use the previous model error as additional information to the system. This is the very idea behind the Error Correction Neural Network (ECNN).

### 3.1 Mathematical Description

A basic dynamical recurrent system depicted in Fig. 3.1 can at time  $t$  be expressed as follows:

$$s_t = f(s_{t-1}, u_t) \quad \text{state transition} \quad (3.1)$$

$$y_t = g(s_t) \quad \text{output equation} \quad (3.2)$$

Functions  $f$  and  $g$  are not specified.  $y_t$  is the computed output and  $s_t$  describes the state. Note that Eq. 3.1 and 3.2 without external inputs  $u_t$  would represent an autonomous system. Let the observed model error at the previous time  $t-1$  act as an additional input to the system. We get ( $y^d$  denotes observed data)

$$s_t = f(s_{t-1}, u_t, y_{t-1} - y_{t-1}^d) \quad (3.3)$$

$$y_t = g(s_t) \quad (3.4)$$

The search for an optimised solution of Eq. 3.3 and 3.4 with respect to functions  $f$  and  $g$  can be stated as

$$\min_{f,g} \frac{1}{T} \sum_{t=1}^T (y_t - y_t^d)^2 \quad (3.5)$$

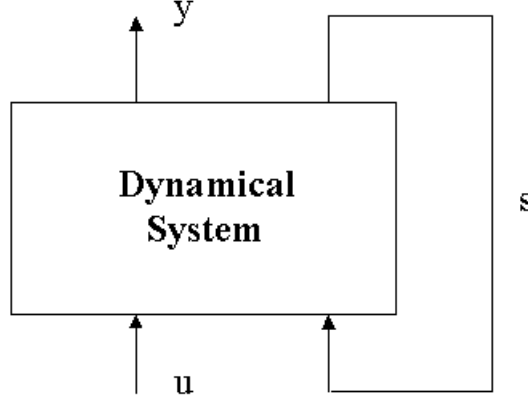


Figure 3.1: A dynamical system with input  $u$ , output  $y$  and internal state  $s$ .

where  $T$  is the number of *patterns*<sup>1</sup>. Implemented as a neural network (denoted  $\aleph$ ) Eq. 3.3 and 3.4 become

$$s_t = \aleph(s_{t-1}, u_t, y_{t-1} - y_{t-1}^d; v) \quad (3.6)$$

$$y_t = \aleph(s_t; w) \quad (3.7)$$

Functions  $f$  and  $g$  have been specified as neural networks with parameter vectors  $v$  and  $w$  respectively. Naturally the optimisation problem of Eq. 3.5 transforms to

$$\min_{v,w} \frac{1}{T} \sum_{t=1}^T (y_t - y_t^d)^2. \quad (3.8)$$

Our next step is to apply an activation function, e.g.  $\tanh(\cdot)$ , and formulate the system of Eq. 3.6 and 3.7 more explicitly [Gro02]. We get

$$s_t = \tanh(As_{t-1} + Bu_t + D(Cs_{t-1} - y_{t-1}^d)) \quad (3.9)$$

$$y_t = C(s_t) \quad (3.10)$$

with weights  $v = \{A, B, D\}$  and  $w = \{C\}$ . Note that a numerical ambiguity arises. Both matrix  $A$  and  $DC$  could code the autoregressive structure of the system. Adding a non-linearity is a measure to avoid this problem [Gro02, p. 83]. This yields

$$s_t = \tanh(As_{t-1} + Bu_t + D \tanh(Cs_{t-1} - y_{t-1}^d)) \quad (3.11)$$

<sup>1</sup>Patterns are data points in a time series.

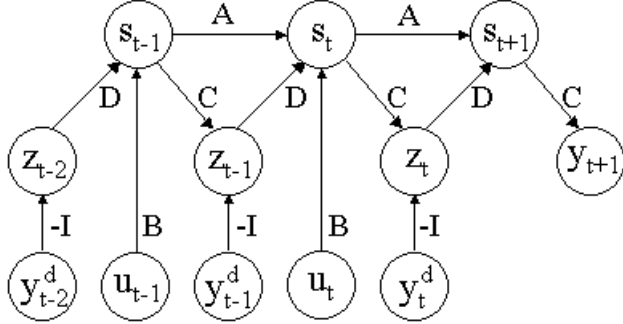


Figure 3.2: The error correction neural network.

$$y_t = C(s_t) \quad (3.12)$$

The parameter optimisation task (cf. Eq. 3.8) with respect to matrices  $A$ ,  $B$ ,  $C$  and  $D$  becomes

$$\min_{A,B,C,D} \frac{1}{T} \sum_{t=1}^T (y_t - y_t^d)^2 \quad (3.13)$$

At this point we can visualise the architecture of the ECNN (Fig. 3.2) from our mathematical description in Eq 3.11 and 3.12. Distinguishing *external inputs*  $u_t$  affecting the state transition  $s_t$  from *target inputs*  $y_t^d$  is important. Note that  $I$  denotes the fixed identity matrix. As a consequence the target values of output clusters  $z_{t-\tau}$ ,  $\tau = 0, 1, 2$ , are zero. Only the difference between  $y_t$  and  $y_t^d$  influences  $s_t$ .

The ECNN offers forecasts based on the modelling of the recursive structure (matrix  $A$ ), the external forces (matrix  $B$ ) and the error correcting part (matrices  $C$  and  $D$ ). The error correcting part can also be viewed as an external input similar to  $u_t$ .

### 3.1.1 Variants and Invariants

Predicting a high-dimensional dynamical system is difficult. A way of reducing the complexity of the task is to separate the dynamics into time variant and invariant structures. One let the system forecast the variants and eventually combine this forecast with the unchanged invariants. This can be done by connecting the standard ECNN (Fig. 3.2) to a compression-decompression network shown in Fig. 3.3. Matrix  $E$  separates variants from invariants while matrix  $F$  reconstructs the dynamics. The actual forecasting is coded in  $G$ . Refer to [ZIM03] for a detailed description.

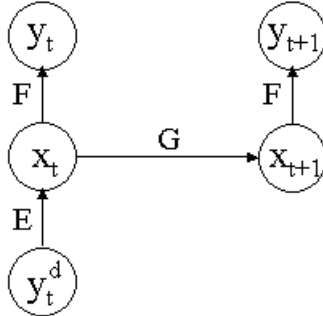


Figure 3.3: Separation of variants and invariants

## 3.2 Training

### 3.2.1 Backpropagation

With a description of the network structure at hand, training matters have to be settled. As previously mentioned the overall objective in training is to minimise the discrepancy between real data and the output of the network. This principle is referred to as *supervised* learning<sup>2</sup>. In a step-by-step manner the error guides the network in the direction towards the target data. The backpropagation algorithm belongs to this class and can be described as “an efficient way to calculate the partial derivatives of the network error function with respect to the weights” [Gro02, p.116]. In Appendix A the algorithm is derived in detail.

### 3.2.2 Learning Algorithm

The backpropagation algorithm supplies information about the gradients. However, a learning rule that uses this information to update the weights efficiently is also needed.

A weight update from iteration  $k$  to  $k + 1$  may look like

$$w_{k+1} = w_k + \eta \cdot d_k \quad (3.14)$$

where  $d_k$  describes the search direction and  $\eta$  the learning rate (or step length). Issues that have to be addressed are how to determine (i) the search direction, (ii) the learning rate and (iii) which patterns to include.

---

<sup>2</sup> The other basic class of learning paradigms is *unsupervised* or *self-organised* learning. One well known learning method in this class is the Self Organising Map (SOM).

A familiar way of determining the search direction  $d_k$  is to apply the *gradient descent* which is a relatively simple rule (see [Hea97]). The major drawback though is that learning easily is caught in a local minima. To avoid this hazard the *vario-eta* algorithm can be chosen as learning rule [NZ98, Sie02]. Basically it is a stochastic approximation of a *Quasi-Newton* method. In the vario-eta algorithm a weight-specific factor,  $\beta^j$ , is related to each weight. For an arbitrary weight, e.g. the  $j$ :th,  $\beta^j$  is defined according to Eq. 3.15.

$$\beta^j = \frac{1}{\sqrt{\sum_{t=1}^N (\frac{\partial E_t}{\partial w^j} - \bar{E})^2}} \quad (3.15)$$

where

$$\bar{E} = \frac{1}{N} \sum_{t=1}^N \frac{\partial E_t}{\partial w^j}.$$

Let us assume there are  $p$  weights in the network. The search direction is determined by multiplying each component of the negative gradient with its weight-specific factor, see Eq. 3.16.

$$d_k = - \begin{pmatrix} \beta^1 & 0 & 0 \\ 0 & \cdot & 0 \\ 0 & 0 & \beta^P \end{pmatrix} \cdot \nabla E \quad (3.16)$$

Above  $E$  denotes the error function and  $N$  the number of patterns. A benefit with the vario-eta rule is that weight increments  $\eta \cdot d_k$  become non-static. This property implies a potentially fast learning phase. See [Gro02] for further discussion.

Concerning a reasonable value of the learning rate  $\eta$ , there is no simple answer. The learning rate is many times determined on an ad hoc basis.

Regarding pattern selection, a stochastic procedure can be used. This simply means that the gradient  $\nabla E^M$  of a subset  $M$  of all patterns at hand are used as an approximation of the true gradient  $\nabla E$  according to

$$\nabla E^M = \frac{1}{|M|} \sum_{t \in M} \nabla E_t \quad (3.17)$$

where  $|M|$  denotes the number of elements of  $M$ .

$M$  can be composed in several ways. In our empirical study we picked, with equal probability, a predefined number of patterns (less than 10 percent of the training data) to represent  $M$ . The gradient  $\nabla E^M$  of Eq. 3.17 was computed and used as input to Eq 3.16. Once all weights were updated, out of the remaining training patterns a new subset was picked for the next iteration. (If recent patterns are considered more significant one may prefer a non-uniform probability distribution where it is more likely to chose a recent pattern [Sie02].)

### 3.2.3 Cleaning

The dilemma of overfitting is deeply rooted in neural networks. One way to suppress overfitting is to assume input data not to be exact (which certainly is the case in the field of financial analysis). The total error of pattern  $t$  can be split into two components associated with the weights and the erroneous input respectively. The corrected input data,  $\tilde{x}_t$ , can be expressed as

$$\tilde{x}_t = x_t + \Delta x_t \tag{3.18}$$

where  $x_t$  is the original data and  $\Delta x_t$  a correction vector. During training the correction vector must be updated in parallel with the weights. To this end, the output target difference i.e. the difference in output from using original and corrected input data has to be known which is only true for training data. Accordingly, the model might be optimised for training data but not for generalisation data because the latter has a different noise distribution and an unknown output target difference. To work around this disadvantage the model  $\tilde{x}_t$  is composed according to

$$\tilde{x}_t = x_t + \Delta x_t - \delta. \tag{3.19}$$

$\delta$  is exactly one element drawn at random from  $\{\Delta x_i, i = 1, \dots, T\}$  (memorised correction vectors from training data). A composition with an additional noise term (Eq. 3.19) benefits from distribution properties which is desirable for generalisation. A detailed motivation is given in [NZ98].

The input modification “cleaning with noise” described above helps the network to concentrate on broader structures in data. To some extent the model is prevented from establishing false causalities.

### 3.2.4 Stopping Criteria

For how many *epochs*<sup>3</sup> should a network be trained? Mainly two paradigms exist, late and early stopping. Late stopping means that the network is trained until a minimum error on the training set is reached, i.e. the network is clearly overfitted. Then different techniques are used to exterminate nodes in the network (known as *pruning*). By doing so eventually a good generalisation ability is reached.

The concept of early stopping is a way of avoiding overfitting. During learning the progression is monitored and training is terminated as soon as signs of overfitting appear. A clear advantage with early stopping is that the time of training is relatively short. On the downside it is hard to know when to stop.

---

<sup>3</sup>An epoch is completed when all training patterns have been read in exactly once.

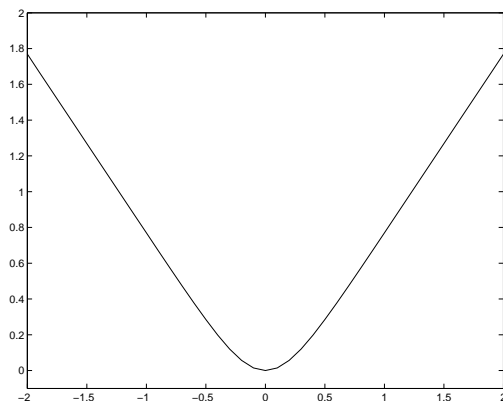


Figure 3.4: The *lncosh* error function of Eq. 3.20,  $a = 3$ .

### 3.2.5 Error Function

When modelling one also has to be aware of *outliers* in data. Outliers typically appear when the economic or political climate is unstable or unexpected information enter the market. By picking an appropriate error function the impact of outliers can be restrained.

The  $\ln \cosh(\cdot)$  error function

$$\frac{1}{a} \ln \cosh(a(o_i - t_i)) \tag{3.20}$$

is a smooth approximation of the absolute error function  $|o_i - t_i|$ .  $o_i$  denotes the response from output neuron  $i$  and  $t_i$  the corresponding target.  $a \in [3, 4]$  has proven to be suitable for financial applications [NZ98].

Compared to a quadratic error function the  $\ln \cosh$  error function has a reminiscent behaviour in the region around zero but not for large positive or negative values as we see in Fig. 3.4. The advantage with this function when modelling financial data is that a large difference in output and target yields a limited and more reasonable error.

# Chapter 4

## Evaluation

A crucial part of financial forecasting is the evaluation of the prediction algorithm. Several performance measures are widely used, but a performance measure in itself is not sufficient for a satisfying evaluation. Relevant benchmarks are also needed. A benchmark is basically a different prediction algorithm used for comparison.

Our intention is to give a few examples of how to judge the quality of a prediction algorithm.

### 4.1 Benchmarks

Any prediction algorithm claiming to be successful should outperform the *naive* predictor defined as

$$\hat{y}_{t+1} = y_t \tag{4.1}$$

where  $y_t$  is the current value of the stock and  $\hat{y}_{t+1}$  the predicted value one time-step into the future. Eq. 4.1 states that the most intelligent suggestion of tomorrow's price is today's price which is a direct consequence of the *Efficient Market Hypothesis* (EMH)<sup>1</sup>. In the empirical study a comparison to the naive prediction of returns was made. The definition is

$$\hat{R}_{t+1} = R_t \tag{4.2}$$

where  $R_t$  is the last known return and  $\hat{R}_{t+1}$  the predicted one-step return.

For a prediction algorithm with incorporated buy and sell signals it could be useful to do a comparison with the *buy-and-hold* return  $R_b$ . This strategy

---

<sup>1</sup>The EMH states that the current market price reflects the assimilation of all information available. Therefore no prediction of future changes in the price can be made given this information [Hel98].

expresses the profit made when making an investment at the start of a time period and selling  $n$  time-steps into the future, i.e.

$$R_b = 100 \cdot \frac{y_{t+n} - y_t}{y_t}. \quad (4.3)$$

A comparison to the buy-and-hold strategy simply gives an indication of the quality of the signals. Is it more profitable to be a “passive” investor?

## 4.2 Performance Measures

Three frequently used measures, namely *hit rate*, *return on investment* and *realised potential*, are defined below. Let us begin with the hit rate  $H_R$ .

$$H_R = \frac{|\{t | R_t^k \hat{R}_t^k > 0, t = 1, \dots, N\}|}{|\{t | R_t^k \hat{R}_t^k \neq 0, t = 1, \dots, N\}|} \quad (4.4)$$

where  $R_t^k$  ( $\hat{R}_t^k$ ) is the actual (predicted)  $k$ -step return<sup>2</sup> at time  $t$ . The norm simply is the number of elements in the series. Eq. 4.4 indicates how often the algorithm produces a correct prediction. In this context a prediction is correct when the direction of the stock  $k$  time-steps into the future is successfully predicted.

The return on investment  $ROI$  takes into account not only the correctness of the sign, but also the quantity of the actual return. The definition is

$$ROI = \sum_{t=1}^T R_t \cdot \text{sign}(\hat{R}_t). \quad (4.5)$$

Finally, in Eq. 4.6 the definition of realised potential  $RP$  is given.

$$RP = \frac{\sum_{t=1}^T R_t \cdot \text{sign}(\hat{R}_t)}{\sum_{t=1}^T |R_t|} \quad (4.6)$$

The realised potential states how large part of the total movement (upwards and downwards) the prediction algorithm successfully identifies.

---

<sup>2</sup>According to the definition the  $k$ -step return  $R_t^k = \frac{y_t - y_{t-k}}{y_{t-k}}$ .

## Chapter 5

# Empirical Study

In the empirical study well traded stocks with a reasonable *spread*<sup>1</sup> were considered. Mainly because a large spread may be fatal if one wishes to incorporate the predictions in real trading. We decided to make one-day predictions (using daily data) of the Swedish stock index (SXGE), Ericsson B and Volvo B with a standard ECNN. Also one-week predictions (using weekly data) of Ericsson B and Volvo B were performed on an ECNN separating variants and invariants (see Sec. 3.1.1). Basically, one assumes that certain time invariant structures can be identified and learned quickly. This means that the latter part of the training is performed with some weights frozen. The occurrence of invariant structures could prove to be more evident in a weekly compared to a daily model, because patterns originate from the same day of the week in a weekly model.

### 5.1 Data Series

For all predictions the following four time series were used as raw input:

- Closing price  $y$  (price of the last fulfilled trade during the day)
- Highest price paid during the day,  $y_H$
- Lowest price paid during the day,  $y_L$
- Volume  $V$  (total amount of traded stocks during the day)

Additionally, external time series served as input. Tab. 5.1 gives a summary of time series considered to have a significant impact on the behaviour of the Ericsson B and Volvo B.

---

<sup>1</sup>The spread is the difference between highest and lowest price paid for the stock during the day.

Regarding the SXGE prediction, another set of external inputs was used. Tab. 5.2 gives the full list.

All data used in the modelling was acquired by Tomlab Optimization AB from the daily service provider *Human Securities AB* (<http://www.humansecurities.se>).

Table 5.1: External time series used predicting Ericsson B and Volvo B.

Stock Model Inputs	
Dow Jones Stock Index	Swedish Stock Index
3-month interest rate Sweden	5-year interest rate Sweden
Swedish SEK / USD FX-rate	Swedish SEK / DEM FX-rate

Table 5.2: External time series used predicting the Swedish stock index SXGE.

Index Model Inputs	
S&P 500	SX 16
Nikkei 225	Gold Price [\$/oz]
3-month interest rate Sweden	5-year interest rate Sweden
Dow Jones Stock Index	German DAX
Swedish SEK / USD FX-rate	Swedish SEK / DEM FX-rate

## 5.2 Technical Considerations

Prior to each training session relevant data series were transformed and preprocessed in different ways. For all external time series (Tab. 5.1 and 5.2) we calculated the normalised one-step return  $R_t$ . The *Gaussian volume* was computed in a running window of 30 days and 12 weeks for daily and weekly predictions respectively. Details are found in Appendix B.

On inputs  $y$ ,  $y_H$  and  $y_L$ , we applied the *log-return*. The definition is given in Eq. 5.1. For small changes the log-return is similar to the one-step return  $R_t$ .

$$R_t^{\log} = \log \frac{y_t}{y_{t-1}}. \quad (5.1)$$

Data were also divided into three subsets: a training set, a validation set and a generalisation set. Roughly, half of all patterns available were used for training and one-quarter each for validation and generalisation. The generalisation period ran over 12 months for both the daily and the weekly model.<sup>2</sup>

<sup>2</sup>In the daily model some training sessions were performed on a shorter generalisation set due to missing data.

As error function we used the *lncosh* function of Eq. 3.20 with  $a=3$ . The standard *tanh*( $\cdot$ ) function served as activation function.

### 5.3 Training Procedure

Weights were initialised randomly (uniformly distributed in the interval  $[-1, 1]$ ). The ECNN was trained (on the training set) for 5000 and 10000 epochs in the daily model and weekly model respectively. We could see that this was enough for the cleaning error (correction to the net inputs) to stabilise.

After training, weights associated with the best performance (in terms of hit rate) on the validation set were selected and applied to the generalisation set to get the final results.

### 5.4 Implementation

Siemens provide a software for neural computing, SENN (Simulation Environment for Neural Networks). SENN is designed to build artificial neural networks based on forecasting and classification models. Prior to running the program, network architecture and details of data (technical indicators, set of patterns etc.) have to be specified in text files. Once these files are loaded into the software it is possible to start training. The graphical interface provides a possibility to monitor this process closely. Continuously data are written to files for post-processing.

# Chapter 6

## Results

In the following two sections, results from the empirical study are presented. Tables and figures are based on generalisation data. The naive prediction of returns constitute benchmark for each prediction model. The prediction of the SXGE is also compared to a similar study of the German DAX-index.

Values of hit rate (HR) and realised potential (RP) using the ECNN and the naive strategy are given in tables below. Graphs of the return on investment (RoI) are also presented (accumulated return over the generalisation set).

### 6.1 Daily Predictions

Table 6.1: Daily predictions of SXGE. Hit rate and realised potential using the ECNN and the naive predictor.

Period	1-day forecast (%)			
	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
1, Jan. 93 to Dec. 93	52.4	10.8	54.7	22.3
2, Jan. 94 to Dec. 94	56.3	17.8	52.0	15.2
3, Jan. 95 to Dec. 95	54.5	11.0	51.4	8.9
4, Jan. 96 to Dec. 96	57.9	24.5	50.0	1.7
5, Jan. 97 to Dec. 97	59.1	32.4	52.8	12.5
6, Jan. 98 to Dec. 98	57.9	24.4	53.5	18.3
7, Jan. 99 to Dec. 99	60.1	26.1	53.9	12.0
8, Jan. 00 to June 00	56.2	21.0	48.5	4.3
mean	<b>56.8</b>	<b>21.0</b>	<b>52.1</b>	<b>11.9</b>

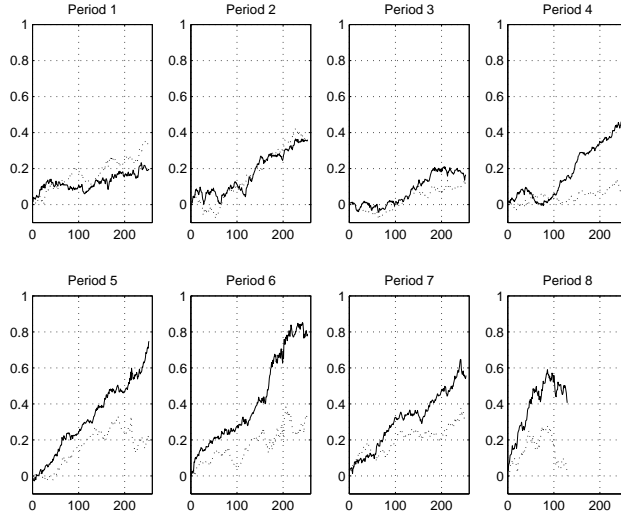


Figure 6.1: Daily predictions of SXGE. Return on investment using the ECNN and the naive strategy, solid and dashed line respectively. Period 1 to 8 are equivalent to the ones presented in Tab. 6.1.

Table 6.2: Daily predictions of Ericsson B. Hit rate and realised potential using the ECNN and the naive predictor.

Period	1-day forecast (%)			
	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
1, Jan. 93 to Dec. 93	47.6	10.1	48.0	22.6
2, Jan. 94 to Dec. 94	51.2	9.3	41.7	2.1
3, Jan. 95 to Dec. 95	55.7	17.8	48.2	4.5
4, Jan. 96 to Dec. 96	51.2	15.2	39.8	-6.4
5, Jan. 97 to Dec. 97	56.3	30.8	43.3	-7.3
6, Jan. 98 to Dec. 98	48.0	0.0	46.5	-0.4
7, Jan. 99 to Dec. 99	53.5	3.7	40.7	-0.5
8, Jan. 00 to Oct. 00	50.7	5.1	42.4	-5.8
mean	<b>51.8</b>	<b>11.5</b>	<b>43.8</b>	<b>1.1</b>

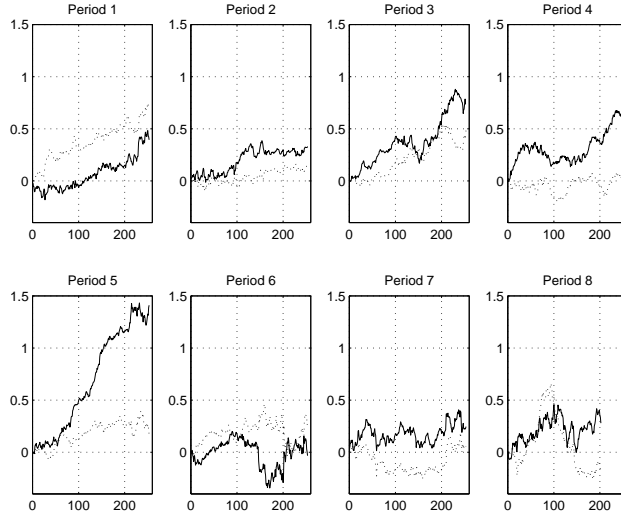


Figure 6.2: Daily predictions of Ericsson B. Return on investment using the ECNN and the naive strategy, solid and dashed line respectively. Period 1 to 8 are equivalent to the ones presented in Tab. 6.2.

Table 6.3: Daily predictions of Volvo B. Hit rate and realised potential using the ECNN and the naive predictor.

Period	1-day forecast (%)			
	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
1, Jan. 93 to Dec. 93	44.9	9.7	41.3	12.8
2, Jan. 94 to Dec. 94	52.4	21.0	45.7	22.3
3, Jan. 95 to Dec. 95	40.3	-2.5	37.9	1.5
4, Jan. 96 to Dec. 96	46.5	13.8	39.0	5.4
5, Jan. 97 to Dec. 97	44.5	1.3	42.5	5.4
6, Jan. 98 to Dec. 98	44.1	-2.5	47.6	18.4
7, Jan. 99 to Dec. 99	44.9	7.8	42.9	10.6
8, Jan. 00 to Oct. 00	47.8	15.0	45.8	12.2
mean	<b>45.7</b>	<b>8.0</b>	<b>42.8</b>	<b>11.1</b>

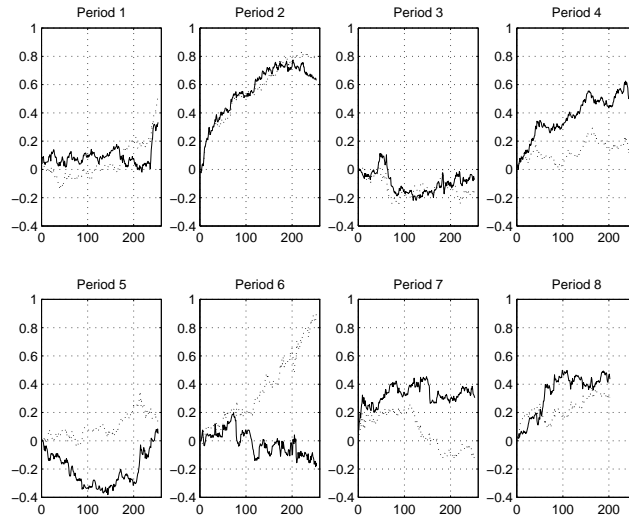


Figure 6.3: Daily predictions of Volvo B. Return on investment for using ECNN and the naive strategy, solid and dashed line respectively. Period 1 to 8 are equivalent to the ones presented in Tab. 6.3.

## 6.2 Weekly Predictions

Table 6.4: Weekly predictions of Ericsson B. Hit rate and realised potential using the ECNN and the naive predictor.

Period	1-week forecast (%)			
	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
1, Jan. 93 to Dec. 93	42.2	-12.9	42.2	-34.1
2, Jan. 94 to Dec. 94	48.9	22.1	44.4	0.3
3, Jan. 95 to Dec. 95	47.8	-8.0	50.0	18.9
4, Jan. 96 to Dec. 96	62.2	25.7	55.6	-19.0
5, Jan. 97 to Dec. 97	51.1	-0.9	60.0	2.9
6, Jan. 98 to Dec. 98	57.8	-6.5	55.6	-1.0
7, Jan. 99 to Dec. 99	60.0	25.9	44.4	-18.8
mean	<b>52.9</b>	<b>6.5</b>	<b>50.3</b>	<b>-7.3</b>

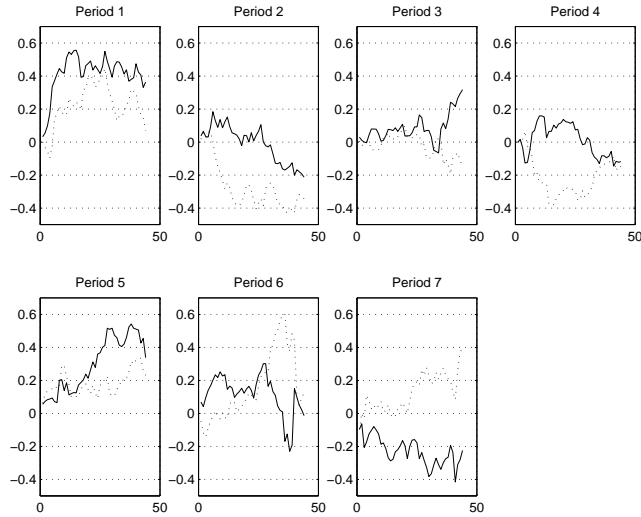


Figure 6.4: Weekly predictions of Ericsson B. Return on investment using the ECNN and the naive strategy, solid and dashed line respectively. Period 1 to 7 are equivalent to the ones presented in Tab. 6.4.

Table 6.5: Weekly predictions of Volvo B. Hit rate and realised potential using the ECNN and the naive predictor.

Period	1-week forecast (%)			
	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
1, Jan. 93 to Dec. 93	57.8	33.4	42.4	-17.8
2, Jan. 94 to Dec. 94	55.6	15.7	60.0	12.5
3, Jan. 95 to Dec. 95	51.1	21.3	40.0	-9.7
4, Jan. 96 to Dec. 96	45.7	13.3	45.7	-17.8
5, Jan. 97 to Dec. 97	66.7	47.7	46.7	12.7
6, Jan. 98 to Dec. 98	60.0	18.0	48.9	-0.5
7, Jan. 99 to Dec. 99	60.0	32.2	44.4	-33.0
mean	<b>56.7</b>	<b>25.9</b>	<b>46.8</b>	<b>-7.7</b>

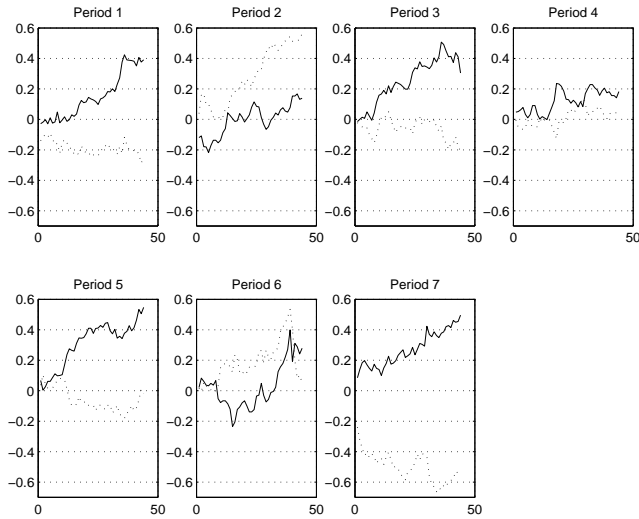


Figure 6.5: Weekly predictions of Volvo B. Return on investment using the ECNN and the naive strategy, solid and dashed line respectively. Period 1 to 7 are equivalent to the ones presented in Tab. 6.5.

### 6.3 Comparison to Benchmarks

As a first observation we see that the ECNN across all performance measures, with some exceptions, is superior to the naive strategy. However, it is worth noticing that the *reverse* naive strategy (i.e.  $\hat{R}_{t+1} = -R_t$ ) could be useful in some cases. Regarding daily data, not surprisingly the prediction of SXGE gave the best results. Looking at Volvo B, the weekly model produces significantly better results compared to the daily model.

In order to get a deeper understanding of the results we made a statistical analysis. Tab. 6.6 and Tab. 6.7 shows the sample standard deviation of the hit rate and realised potential based on data from Sec. 6.1 and Sec. 6.2 respectively. It appears that the ECNN and the naive strategy show similar characteristics in terms of stability.

Table 6.6: The standard deviation of the ECNN and the naive predictor based on daily predictions.

Standard deviation daily predictions (%)				
Stock	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
SXGE	2.5	7.5	2.1	6.9
Ericsson B	3.2	9.8	3.3	9.7
Volvo B	3.5	8.6	3.4	7.0
mean	<b>3.1</b>	<b>8.6</b>	<b>2.9</b>	<b>7.9</b>

Table 6.7: The standard deviation of the ECNN and the naive predictor based on weekly predictions.

Standard deviation weekly predictions (%)				
Stock	$HR_{ECNN}$	$RP_{ECNN}$	$HR_{naive}$	$RP_{naive}$
Ericsson B	7.3	17.3	6.9	17.7
Volvo B	6.8	12.4	6.5	17.0
mean	<b>7.1</b>	<b>14.9</b>	<b>6.7</b>	<b>17.4</b>

To evaluate the results of the SXGE further, we made a comparison to a forecast of the German DAX-index by Zimmermann and Weigend (1997) [ZW97]. The prediction task is similar to the one presented in this thesis with the exception that it was performed on a 6-layer feed-forward network. The authors report hit rates in the region of 55 % for predictions one day ahead. The corresponding result in this thesis of 56.8 % is slightly higher. The deviation is too small for any conclusions to be drawn, but the difference in network design certainly plays part.

# Chapter 7

## Discussion

Maybe the most crucial issue of stock prediction is how to get stability over time. In this respect neither the daily nor the weekly model is not optimised and refinements have to be made. Nevertheless, there is no doubt about neural networks potential in a trading environment. As we have seen in the previous section, the ECNN occasionally shows good results.

Intuitively, one may think that the weight initialisation phase is decisive for the outcome of the predictions. Theoretically, this should not be the case though due to the stochasticity in the selection of patterns during training (see Sec. 3.2.2). To confirm this notion we set up a training scheme where the net was initialised with the “best” (in terms of hit rate) generalisation weights from the previous training period. The results gave indications of an even negative impact (on the hit rate) using a biased initialisation compared to a random one.

This phenomena typically illustrates the challenge we face when trying to validate a dynamical system for a longer period of time. Previously gained information about the system may be hard to utilise successfully in a future time perspective.

### 7.1 Possible Improvements

Let us conclude this thesis with some thoughts on how the work presented may be extended.

To this point the network has been tested on a few different stocks only. Obviously, the ECNN should be validated on several more stocks over different time periods on both daily and weekly data.

Regarding performance measures some refinements might be fruitful. For example, a prediction may be considered a hit only if the sign is correctly predicted

*and* a certain threshold level is reached. In this case it is more unlikely that noise itself triggers a hit. Naturally the credibility of the indicator is strengthened. In the same manner the realised potential can be redefined.

A further development is to combine one-day predictions with multi-day predictions. To estimate the relevance of a possible detected trend a weighted sum could be calculated, where weights are associated with the one-day forecast, two-day forecast and so on in descending order. Eventually, the aggregate is used to make an assessment of whether the trend is likely to turn out as predicted or not

Moreover, the forecasts of a committee of models could be averaged. This will bring about a reduction of the forecast variance, thus a more reliable output, also suggested in [Moo98].

For trading purposes, a ranking system among different stocks may be thought. With a reliable model at hand several stocks can be trained simultaneously. At the end of a training session the results are assessed to performance measures, general status etc. Eventually, the stock(s) with the highest rank will be considered for trading. On a regular basis the network is retrained and a new ranking list is compiled.

In a larger perspective, several different models *and* stocks might be utilised in parallel. As for the ranking list, the aim is to identify the currently most suitable stock. Stating and solving an optimisation task where all conceivable factors are taken into account will produce the final output. Ultimately, computational costs have to be put in relation to potential improvements.

A different angle of improving the ECNN is to focus on the characteristics of stocks predicted. By associating trends, volumes, volatility etc. to the performance of the network, valuable information might be learned. Maybe it would be possible to distinguish a group of stocks (in terms of technical data) that the ECNN forecasts well.

# Bibliography

- [Ati00] Atienza F. et al. *Risk Stratification in Heart Failure Using Artificial Neural Networks*, Research paper, Cardiology Department, University General Hospital, Valencia, 2000. Available at <http://www.amia.org/pubs/symposia/D200367.PDF>
- [BD02] Brockwell P.J. and Davis R.A. *Introduction to Time Series and Forecasting*, 2nd edition, Springer, New York, 2002.
- [Fau94] Fausett L. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*, Prentice-Hall, New Jersey, 1994.
- [Gro02] Grothmann R. *Multi-Agent Market Modeling based on Neural Networks*, Ph.D. thesis, Faculty of Economics, University of Bremen, Germany, 2002.
- [Hay94] Haykin S. *Neural Networks. A Comprehensive Foundation*, Macmillan College Publishing, New York, 1994.
- [Hea97] Heath M.T. *Scientific Computing— An Introductory Survey*, McGraw-Hill, New York, 1997.
- [Hel98] Hellström T. *A Random Walk through the Stock Market*, Licentiate Thesis, Department of Computing Science, Umeå University, Sweden, 1998.
- [KK94] Keller P.E. and Kouzes R.T. *Gamma Spectral Analysis via Neural Networks*, Research paper, Environmental Molecular Sciences Laboratory, Richland, Washington, 1994. Available at <http://www.emsl.pnl.gov/proj/neuron//papers/keller.nss94.pdf>
- [Moo98] Moody J. *Forecasting the Economy with Neural Nets: A Survey of Challenges and Solutions*. In Orr G.B. and Müller K.-R. (Eds.) *Neural Networks: Tricks of the Trade*, pp. 347–371, Springer, Berlin, 1998.
- [NZ98] Neuneier R. and Zimmermann H.-G. *How to Train Neural Networks*. In Orr G.B. and Müller K.-R. (Eds.) *Neural Networks: Tricks of the Trade*, pp. 373–423, Springer, Berlin, 1998.

- [PG02] Palacios R. and Gupta A. *Training Neural Networks for Reading Handwritten Amounts on Checks*, Working paper 4365-02, MIT Sloan School of Management, Cambridge, Massachusetts, 2002. Available at [http://ssm.com/abstract\\_id=314779](http://ssm.com/abstract_id=314779)
- [Sie02] *SENN User Manual. Version 3.1*, Siemens Business Service, Germany, 2002.
- [WZN96] Weigend A.S., Zimmermann H.-G. and Neuneier R. *Clearning*. In Refenes A.-P. et al. (Eds.) *Neural Networks in Financial Engineering. Proc. of the 3rd Int. Conf. on Neural Networks in the Capital Markets*, pp. 511-522, World Scientific, Singapore, 1996.
- [ZGN00] Zimmermann H.-G., Grothmann R. and Neuneier R. *Forecasting High-Dimensional Dynamical Systems by Error Correction Neural Networks with a focus on the German Yield Curve*, Research paper, Siemens AG, Munich, 2000.
- [Zim03] Zimmermann H.-G. *System Identification & Forecasting by Neural Networks. Principles, Techniques, Applications*, Compilation of material presented to a conference held at Mälardalen University, September 2003.
- [ZW97] Zimmermann H.-G., Weigend A.S. *Representing Dynamical Systems in Feed-Forward Networks: A Six Layer Architecture*. In Weigend A.S., Abu-Mostafa Y. and Refenes A.-P. (Eds.) *Decision Technologies for Financial Engineering. Proc. of the 4th Int. Conf. on Neural Networks in the Capital Markets*, pp. 289-306, World Scientific, Singapore, 1997.

## Appendix A

# The Error Backpropagation Algorithm

The derivation of the error backpropagation algorithm is performed on a feed-forward network. The extension to more complex networks is straightforward. Refer to [Gro02] for a deeper review.

Assume a 3-layer feed-forward network (fully connected) with  $l$  neurons in the input layer,  $m$  neurons in the hidden layer and  $n$  neurons in the output layer. Let  $w_{ij}$  denote a connection between the input layer and the hidden layer. Analogous  $w_{jk}$  represents a connection between the hidden layer and the output layer (see Fig. A.1).

$T$  patterns are available for training. After presentation to pattern  $t$  the accu-

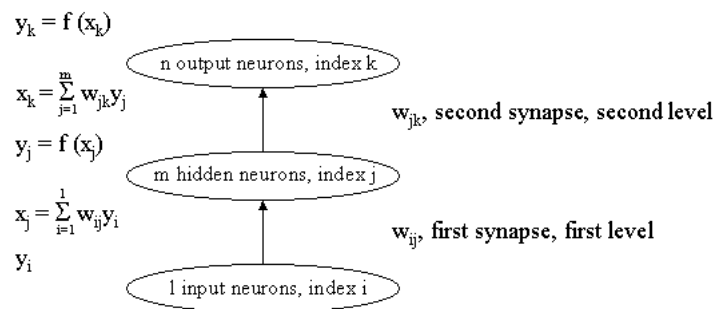


Figure A.1: Information flow in the forward path of a 3-layer neural network.

mulated error signal of the output neurons is

$$E(t) = \frac{1}{2} \sum_{k=1}^n (y_k(t) - y_k^d(t))^2 \quad (\text{A.1})$$

where  $y_k$  and  $y_k^d$  denotes actual and target output respectively. A factor 0.5 is included for computational reasons. Consequently, the average error of an arbitrary pattern is

$$E_{av} = \frac{1}{T} \sum_{t=1}^T E(t). \quad (\text{A.2})$$

The overall objective is to minimise  $E_{av}$  by adjusting the free parameters  $w_{ij}$  and  $w_{jk}$ . To achieve this, first we have to calculate the partial derivatives of Eq. A.2 with respect to the weights  $w_{jk}$ . We get

$$\begin{aligned} \frac{\partial E_{av}}{\partial w_{jk}} &= \frac{1}{T} \sum_{t=1}^T (y_k - y_k^d) \frac{\partial y_k}{\partial x_k} \frac{\partial x_k}{\partial w_{jk}} \\ &= \frac{1}{T} \sum_{t=1}^T (y_k - y_k^d) f'(x_k) y_j. \end{aligned} \quad (\text{A.3})$$

In Eq. A.3 the *chain-rule*<sup>1</sup> was used to compute the partial derivatives. Let

$$\delta_k = (y_k - y_k^d) f'(x_k). \quad (\text{A.4})$$

The substitution of  $\delta_k$  in Eq. A.3 yields

$$\frac{\partial E_{av}}{\partial w_{jk}} = \frac{1}{T} \sum_{t=1}^T \delta_k y_j. \quad (\text{A.5})$$

Since the network is assumed to be fully connected a set of  $k \cdot j$  partial derivatives exists. These derivatives (Eq. A.5) constitute the cumulative gradient of the second level in the network (see Fig. A.1).

In similar fashion the partial derivatives of Eq. A.2 with respect to  $w_{ij}$  are calculated (the first level of the network). Using the chain-rule twice leads to

$$\begin{aligned} \frac{\partial E_{av}}{\partial w_{ij}} &= \frac{1}{T} \sum_{t=1}^T \sum_{k=1}^n (y_k - y_k^d) \frac{\partial y_k}{\partial x_k} \frac{\partial x_k}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{ij}} \\ &= \frac{1}{T} \sum_{t=1}^T \sum_{k=1}^n (y_k - y_k^d) f'(x_k) w_{jk} f'(x_j) y_i. \end{aligned} \quad (\text{A.6})$$

---

<sup>1</sup>If  $y = f(g(x))$  then  $\frac{dy}{dx} = \frac{dy}{dg} \frac{dg}{dx}$ .

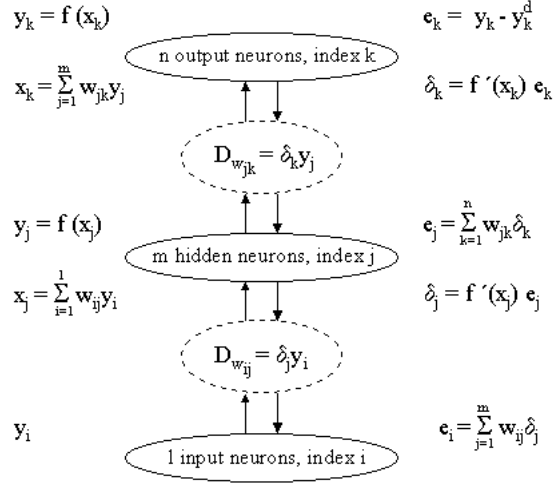


Figure A.2: Error backpropagation of a single training pattern. Error  $e_k$ ,  $e_j$  and  $e_i$  induce the actual error flow.

Substituting the auxiliary term  $\delta_k$  in Eq. A.6 yields

$$\frac{\partial E_{av}}{\partial w_{ij}} = \frac{1}{T} \sum_{t=1}^T \sum_{k=1}^n \delta_k w_{jk} f'(x_j) y_i. \quad (\text{A.7})$$

For simplicity an additional auxiliary term is defined according to

$$\delta_j = f'(x_j) \sum_{k=1}^n w_{jk} \delta_k. \quad (\text{A.8})$$

By substituting  $\delta_j$  in Eq. A.7 the cumulative gradient of the first network level can be written

$$\frac{\partial E_{av}}{\partial w_{ij}} = \frac{1}{T} \sum_{t=1}^T \delta_j y_i. \quad (\text{A.9})$$

To summarise the backpropagation algorithm let us consider a single training pattern. On the forward path the outputs  $y_i$  and  $y_j$  are calculated (see Fig. A.2). On the backward path the partial derivatives on level one and two appear as the products  $\delta_j y_i$  and  $\delta_k y_j$  respectively. In other words, the error information is implicitly carried through the network by the auxiliary terms  $\delta_j$  and  $\delta_k$  (Eq. A.4 and A.8).

## Appendix B

# Preprocessing

The one-step return  $R_t$  is defined as the relative increase in price since the previous point in the time series, i.e.

$$R_t = \frac{y_t - y_{t-1}}{y_{t-1}}. \quad (\text{B.1})$$

$R_t^n$  gives the normalised one-step return:

$$R_t^n = \frac{R_t - \bar{R}}{\sigma} \quad (\text{B.2})$$

where  $\bar{R}$  is the mean and  $\sigma$  the standard deviation of the (return) time series.

The definition of the *Gaussian volume*  $V_n^G(t)$  is

$$V_n^G(t) = \frac{V(t) - m_v(t)}{\sigma_v(t)} \quad (\text{B.3})$$

where

$$m_v(t) = \frac{1}{n} \sum_{i=1}^n V_{t-i} \quad (\text{B.4})$$

and

$$\sigma_v(t) = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (V(t-i) - m_v(t))^2}. \quad (\text{B.5})$$

$n$  is the length of the running window.