Forecasting Economic Time-Series with Neural Networks

KLAUS KÖHLER*

1. INTRODUCTION

Forecasting the behaviour of economic systems, such as national economies and stock or trade markets, represented by indexes, differs from predicting the behaviour of technical systems in a way that the underlying laws are usually not well understood. This holds particularly true for short term predictions where fundamental relations gain little influence. If the underlying laws are unknown, one can try to empirically find regularities of the system by observing its behaviour in the past.

Most classical methods of forecasting, e.g. linear regression or autoassociative integrated moving average (ARIMA), use linear models. But economic relations are usually highly nonlinear and recurrent. Therefore, forecast results of economic indexes are usually biased, less accurate, and afflicted with a greater variance than those of technical systems.

On the other hand, even rather imprecise forecasts of economic data may be valuable. Predicting the trend (up/down) of the exchange rate between the $US and the DM for the next day with a correctness of 61% [Zim 91] may enable banks to achieve high profits. Moreover, long term predictions can serve as a valuable basis for decisions concerning investments or even fiscal policy.

Neural networks can overcome or at least alleviate some of the problems inherent in economic predictions since they can build nonlinear models from observed behaviour of systems.

In paragraph 2 the properties of neural networks are described and linked to the task of forecasting economic time-series in paragraph 3. Paragraph 4 is devoted to backpropagation networks. In paragraph 5 forecasting problems, specific to economic questions, are discussed. Solutions to these problems are proposed in the main paragraph 6, where several methods

* Department of Computer Science, Fachhochschule München, FRG
1 DAX = Deutscher Aktienindex = German stock index.
of tackling the problem of generalization are discussed in detail. These methods are applied to the task of predicting the DAX one week in advance and the results are presented in paragraph 7 and compared with results obtained from linear regression. The concluding paragraph 8 summarizes the advantages of and problems with using backpropagation networks for economic time-series prediction.

2. GENERAL PROPERTIES OF NEURAL NETWORKS

Most statistical methods of time-series forecasting depend on assumptions about the underlying laws of a given phenomenon. Parametric models allow to adapt only a few parameters. Often these models use linear relations between the observable (or transformed) input data and the output data, e.g. the simplest autoregressive model

$$y(t+1) = \beta + \sum_{i=0}^{1} \alpha_i y(t-i)$$

with real parameters $\beta$ and $\alpha_i$.

Neural networks have a great number of adaptable parameters, called weights, and can be considered as nonparametric models. They provide a higher degree of freedom and some types can approximate any continuous function describing the input/output behaviour of a system. In particular, they use nonlinearities to model the input/output relations. The huge number of parameters are adjusted through a training process.

The backpropagation network described in the next paragraph uses supervised learning to adapt its weights. This means that past input/output data pairs are presented to the network which thereupon adapts its weights until its output calculated from the presented input is almost identical to the target output. Thus the model selection is mainly done by the network itself and the user need not know the laws which govern the system or the type of function describing its input/output behaviour.²

3. PATTERN ASSOCIATION IN ECONOMIC TIME-SERIES FORECASTING

Predicting the future on the basis of observed regularities means to find patterns in the time-series. We can distinguish time patterns from space

²This property and its negative implications share neural networks with other nonparametric methods.
patterns. Time patterns are regularities in a single time-series, e.g. V-, W-, M-, and headshoulder-formations in the technical analysis (chart analysis) of stock prices. Time patterns of a time-series \( y \) can be modelled by a function

\[
y(t + 1) = f(y(t - l), \ldots, y(t - 1), y(t)).
\]

Space patterns relate different time-series. They can be modelled by a function

\[
y = f(a, b, c, \ldots),
\]

e.g.

\[
DAX = f(capacity\ utilization,\ outstanding\ orders,\ interest\ rate,\ put-call-volume, \ldots).
\]

Of course, the predicted value may also be a vector, e.g.

\[
y = (DAX(t + l), DAX(t + 2), up/down(DAX(t)), \ldots)
\]

In this case an input pattern has to be associated with an output pattern. There is an ever lasting dispute between fundamentalists who believe that most economic time-series can only be effectively forecast on the basis of other time series representing fundamental data, and the more technically inclined chart analysts who assume that all external factors are reflected automatically in the time series and therefore only look for time patterns. In the following we shall use both space and time patterns, e.g.

\[
DAX((t + 1) = f(DAX(t - 4), \ldots, DAX(t), FAZ-Frühindikator(t), interest\ rate(t - 20), \ldots)\]

window of length 5 lag 20

The neural net can decide itself what data to use. This will give us some hints concerning the importance of the data for the respective prediction.

4. USING BACKPROPAGATION NETWORKS FOR PATTERN ASSOCIATION

The most versatile artificial neural network, backpropagation, which was the basis of our research on time-series forecast, will be presented in a rather informal way to establish a common language and common notations.

\(^{1}\) FAZ-Frühindikator = early indicator of the Frankfurter Allgemeine Zeitung.
4.1. Pattern Association with Feedforward Networks

Artificial neural networks (ANN) are designed to mimic natural neural nets. Both consist of a great number of simple neurons which can work parallel. Each neuron is connected with many other neurons. A neuron has an activity level which determines its output. The output signal is communicated to other neurons via synapses which can amplify or diminish it. This is simulated in ANNs by multiplying the output signal with the weight of the connection. All incoming weighted signals are added and the sum is compared with an individual threshold of the neuron. The difference, which is called net input, is passed through an activation function or transfer function to produce the activation level of the neuron.

Usually a sigmoidal squashing function, e.g. the hyperbolic tangent, is used for activation. It can be interpreted as a differentiable approximation of the sign function.

FIGURE 1.

![Artificial Neuron and Layered Feedforward Network Diagram](image)

Neurons with common properties are clustered. In feedforward networks the clusters which are called layers build a hierarchy. The neurons within a layer are not connected with each other but with some or all neurons of adjacent layers. An input layer serves as buffer for an input pattern. Its activation function is the identity. Activity patterns are propagated synchronously from the input layer(s) via the hidden layer(s) to the output layer(s).

An input pattern, e.g. a window \((y(t-l), \ldots, y(t))\) of a time series is mapped onto an output pattern, e.g. a vector \(o = (o_1, o_2)\) that is to predict the target window \((y(t+l), y(t+2))\) of the time series.

In principle, a feedforward net with a single hidden layer of sufficient

\(^4\)The threshold \(b_i\) is realized by an additional weight \(\theta_i = -b_i\) from a common bias neuron with the constant activation 1.
size and sigmoidal squashing function can approximate any continuous mapping of input patterns to output patterns. But in practice it is hard to find appropriate weights to realize this mapping. Additional hidden layers can sometimes simplify the task of weights adaptation.

4.2. Learning in Backpropagation Networks

The weights of a feedforward net have to be adjusted so that the desired mapping

\[ x^p \rightarrow t^p, \ p = \text{pattern index} \]

is approximated by the network input/output function

\[ f : x^p \rightarrow o^p \]

i.e. the global error

\[ E = \sum_p E^p \]

has to be minimized. Here

\[ E^p = \frac{1}{2} \sum_j (t_j^p - o_j^p)^2, \]

where \( j \) runs through all output neurons, is the error of pattern \( p \) and \( t_j^p - o_j^p \) is the error of pattern \( p \) at the output neuron \( j \).

For feedforward networks without hidden layers weight adaptation is a simple task. The weights are initialized with small random values. Then the
pairs \((x^p, o^p)\) of input and target patterns are presented to the network one after the other. The output value of the net is calculated and compared with the target value. Then the weights between input and output layer are adapted according to the rule

\[
w_{ji} := w_{ji} + \Delta w_{ji}, \quad \Delta w_{ji} = \sum_p \Delta w^p_{ji}, \quad \Delta w^p_{ji} = \eta \cdot \delta^p_j \cdot o^p_i, \quad \delta^p_j = t_j^p - o_j^p
\]

The weight change \(\Delta w^p_{ji}\) is proportional to the error \(\delta^p_j\) and to the contribution \(o^p_i\) of the input neuron \(i\) to the error. The learning rate \(\eta\) is a small positive real number: \(0 < \eta < 1\).

It can easily be shown that the delta rule

\[
w_{ji} := w_{ji} + \Delta w_{ji}, \quad \Delta w_{ji} = \sum_p \Delta w^p_{ji}, \quad \Delta w^p_{ji} = \eta \cdot \delta^p_j \cdot o^p_i, \quad \delta^p_j = (t_j^p - o_j^p) \cdot F'(net_j^p)
\]

describes a steepest descent to the nearest «valley» of the global error function. For feedforward networks with hidden layers the weights to output neurons can be adapted in the same way. The delta rule can also be used to adapt the weights to the (next) hidden layer but it is not obvious how the error \(\delta^k_k\) at the hidden neuron \(k\) can be determined.

**Figure 3.**

- Error at output neuron: \(\delta^p_j = (t_j^p - o_j^p) \cdot F'(net_j^p)\)
- Weight change: \(\Delta w_{jk}^p = \eta \cdot \delta^p_j \cdot o^p_k\)
- Error at hidden neuron: \(\delta^p_k = (\sum_j \delta^p_j \cdot w_{jk}) \cdot F'(net_k^p)\)
- Weight change: \(\Delta w_{ki}^p = \eta \cdot \delta^p_k \cdot o^p_i\)

A simple mathematical analysis [Rum86] yields

\[
\delta^p_k = (\sum_j \delta^p_j \cdot w_{jk}) \cdot F'(net_k^p).
\]

\(^5\) For small absolute values of \(net\), the derivative \(F'(net)\) is nearly 1 if \(F(x) = \tanh(x)\), so that the delta-rule coincides with the simplified version given above. A weight \(w_{ji}\) is changed less if the net input \(net_i\) of its target neuron increases and thus its activation approaches saturation.
The errors $\delta_j$ at the output neurons $j$ are backpropagated to the hidden layer. A hidden neuron $k$ with a large weight $w_{jk}$ contributes to the error $\delta_j$ to a higher degree than a hidden neuron with smaller weight. Therefore, it is assigned a greater credit $\delta_j \cdot w_{jk}$. The contributions $\delta_j \cdot w_{jk}$ of neuron $k$ to the errors $\delta_j$ are added giving the error $\delta_j^2$.

If further hidden layers exist, their errors are calculated successively in the same way according to the errors of the layers above. The patterns are repeatedly presented to the network and the weights are adapted until the global error has become sufficiently small$^6$.

5. DIFFICULTIES IN FORECASTING TIME SERIES

Besides the general problems of model selection there are problems specific to the analysis of time-series with neural networks. Many of these problems will also be encountered when using statistical forecasting methods.

5.1. General Forecasting Problems

Any prognosis is built on the assumption that there are functional relations between certain known data and the prognosis data. These relations must be stable over a longer period of time. This allows for finding the relations through analysis of past data and to extrapolate them with regard to the future. The following questions have to be considered for any forecasting problem.

- **Which data are relevant**: fundamental or technical data or psychological aspects?
  Only specialists in the specific field of economy may answer this question. Usually one gets ten different answers if one asks ten specialists. This does not mean that any of them is wrong. It often happens that different data carry the same information. Thus all time-series named by specialists can be taken into account. This leads to more parameters (bigger neural nets), which implies generalization problems that will be discussed in the next section.

- **Which type of functions connects input data with prognosis data?** Can

$^6$ We use pattern by pattern learning, i.e. $\Delta w_i = \Delta w_i^p$, which has proved to work more efficiently than batch learning and reduces the chance of getting stuck in a local minimum of the global error function.
a simple linear relation be assumed or is the relation more complicated so that nonlinear models have to be used? Economic problems usually involve many factors which cannot be separated because they influence each other. High dimensional problem spaces can be treated with algebraic means if the functions involved are linear. Unfortunately many economic relations are not linear. Nonlinear relations can be treated with calculus but then only a few variables may be involved to keep the calculations manageable. Neural networks open a way to handle nonlinear relations among many variables.

- **Is it possible to describe the relations by a mathematical function or by rules?** Or is no quantitative relation known? If a mathematical model describing the relations between input and prognosis data precisely enough is available, the problem with forecasting is solved and neural networks should not be applied. If rules are known, classical expert systems which use inference mechanisms should be used. If the rules are incomplete or fuzzy, fuzzy logic can be applied, possibly in combination with expert systems.

  If one is pretty sure that there is a functional relation between the selected input data and the predicted data but cannot formulate rules or does not know the function type, then neural nets become an interesting tool of prediction. In section 6.6 we will also discuss briefly how to incorporate knowledge into neural networks.

- **Are the functional relations stable over a period of time,** at least for the forecasting horizon? Did structure breaks often occur in the past? Stability with regard to time is a prerequisite for every sensible prognosis, but it cannot be taken for granted. The stability question can only be answered after a model has been built that can be checked against reality. Methods of evaluating and validating the forecast results and the risk of misinterpretation are discussed in 6.4.

- **Are there factors that might influence the prognosis which cannot be quantified,** such as rumours or psychological aspects? Sometimes psychological factors, such as expectations, are reflected in time series at hand, e.g. put/call indices for stock markets. The «formations» in stock price time series often reflect expectations of the customers not necessarily supported by measurable facts. Such expectations can become self-fulfilling prophecies. If facts which cannot be measured directly are not contained in readily
available time series, they have to be treated as noise. The prognosis variable is interpreted as a stochastic variable

\[ y(t) = d(t) + s(t), \quad d = \text{deterministic part (structure)}, \]
\[ s = \text{stochastic part (noise)} \]

The noise may considerably increase the variance and the size of the confidence interval of the prognosis variable \( y(t) \).

- **Does the noise dominate the structural relation?**
  - If the noise dominates, the model used to describe the economic system is inappropriate. The structural relations between input and output data will probably not be found and the predicted results will become unacceptable. Unfortunately, it is impossible to decide whether bad prognoses stem from an inappropriate model, e.g. irrelevant input data, or unstable relations.

- **Are enough data available** to spot structure and distinguish it from noise?
  - Nonparametric models need much more data to find structure in the data than parametric ones that assume a fixed type of relation. It is one of the great challenges to produce good forecasting results with neural networks with relatively few training data. Because of the huge number of adaptable weights and the resulting flexibility neural nets tend to learn the training examples by heart, thus generalizing poorly. This generalization problem will be discussed in detail in the next section.

### 5.2. The Generalization Problem

If a neural net is big enough, it will learn all the training examples perfectly, thus reducing the global error to zero (as long as there are no contradicting training examples, i.e. the input/output relation is a function in the mathematical sense). Then both the structure and the noise have been learned. Since economic data usually contain much noise and the noise is random by definition, the neural net will behave poorly on unseen data because it predicts the noise. Interpolating or extrapolating to unseen data is called **generalization**. Generalization effectivity is the goal of forecasting. But generalization effectivity decreases as the performance on trained data increases. This is qualitatively shown in the graphic on the right. Actually, generalization performance does not only depend on the number of weights alone but on many other factors as well, e.g. the topology of the network, the magnitude of the weights, etc. Generally speaking,
generalization performance decreases as net complexity increases above a certain level. Therefore, we have to compromise between contradicting goals:

- **complex nets** with many input time series and many hidden neurons in order to be able to find complex relations between input and output data and large weights in order to use the nonlinearity of the activation functions,
- **small and simple nets** to avoid overadaptation and poor generalization performance.

**Figure 4.**

Error

Network complexity

6. **How to Find a Backpropagation Net with Optimal Generalization Performance**

In order to obtain a neural net with good generalization performance we try to keep the net

- as complex as necessary to find all structural relations between input and prognosis data
  and
- as simple as possible so that the noise in the data cannot be learned. In other words, we try to exploit the advantages of parametric models without losing the flexibility of nonparametric models.

There are several design issues which determine the complexity of the resulting network:

- input and target data
- preprocessing and postprocessing of the data
• topology of the net
• training strategies\(^7\)
• error function to incorporate knowledge into the network.

6.1. Selecting input and output data

There is little choice in selecting the output data as they are determined by the problem of prognosis. Design decisions concern

• forecasting horizon
• quantitative aspects, e.g. prediction of the exact stock price (real interval) or only discrete values like up/down or turning points. The first constitutes a regression system, the second a classification system.

The selection of the input data is up to the network designer as far as the data are available\(^8\). The more time series are provided, the more structure may be contained in the data. But the number of data determines the number of input neurons and by it the net complexity. In a simple feedforward net with only one hidden layer the number of weights is roughly proportional to the number of input neurons since there are usually only a few output neurons so that the hidden output connections can be neglected compared to the input-hidden connections. From this point of view, obviously redundant time series should be omitted.

It is easy to check whether there is a linear relation between the input data by determining the correlation matrix. If two time series are highly correlated \(|p|>0.95\), one of them can be dropped. Further reductions of the number of input time series on the basis of linear dependence are briefly discussed in the next section on preprocessing.

6.2. Preprocessing and Postprocessing

There are several reasons for preprocessing and postprocessing the input and output data, respectively:

• Normalization

A neural network whose output neurons use the hyperbolic tangent as activation function can only produce results between -1 and 1.

\(^7\) We do not discuss issues like learning rate, momentum term, second order learning algorithms etc.

\(^8\) One should not underestimate the costs of obtaining relevant data. They may be prohibitive.
Though the input data could in principle be any real values whatever, they should also be mapped onto the real interval \([-1, 1]\). By this means, as long as the weights are small, the net input of the hidden neurons does not become very large and the hidden neurons can work in the linear range of the activation function.

Moreover, through a transformation onto a common interval all input neurons get the same chance to determine the output. Without normalization input data varying in a range of, say, \(-10^{-5}\) to \(10^{-5}\), would be dominated by a series varying between \(10^5\) and \(10^6\).

The transformation function should neither destroy similarities between data nor introduce new similarities because neural networks map similar inputs onto similar outputs\(^9\). If the neural net is to predict whether a value will increase or decrease, coding the distinction with -1 and +1 is a problematic decision. If for example

\[
\begin{align*}
\gamma(t) &= 400.1, \\
\gamma(t + 1) &= 400.0, \\
\gamma(t + 2) &= 400.1, \\
\gamma(t + 3) &= 500.0
\end{align*}
\]

the corresponding target values are

\[
\text{sign}(\gamma(t + 1) - \gamma(t)) = -1, \\
\text{sign}(\gamma(t + 2) - \gamma(t + 1)) = 1, \\
\text{sign}(\gamma(t + 3) - \gamma(t + 2)) = 1.
\]

The first values of the time-series are similar, the last is not. The first two coded target values, however, are very different, whereas the last are identical.

The following transformations are commonly used:

\[
y = \frac{x - m}{s} \quad \text{with} \quad m = \frac{\text{max} + \text{min}}{2} \quad \text{and} \quad s = \frac{\text{max} - \text{min}}{2} \quad \text{linear transformation}^{11}
\]

and

\[
y = \tanh\left(\frac{x - m}{s}\right) \quad \text{with} \quad m = \text{mean value} \quad \text{and} \quad s = \text{standard deviation of } x.
\]

The second transformation reduces the influence of outliers.

\(^9\) A smaller interval, e.g. \([0.9, 0.9]\) is preferable, especially for the output neurons, because the hyperbolic tangent never reaches \(\pm 1\) so that the error will never become zero.

\(^{10}\) We do not discuss the issue of similarity here. For real valued economic time series the euclidean distance is usually an appropriate similarity measure. This does not necessarily hold for periodic series. If, for example, one input neuron is supplied with the day of the week coded with 1, 2, ..., 7, then 7 is a neighbour of 1. For an extensive discussion of the similarity issue, see [Koh87].

\(^{11}\) Some neural network simulators provide this pretransformation as a built-in feature.
• Reduction of the number of neurons

If two time-series are highly correlated, one of them can be dropped. But even if several time-series are mutually nearly uncorrelated, it may happen that their combination contains redundant information. To reduce linear dependence one can select the principal components [HKP91] only, i.e. the linear combinations of time series which are eigenvectors of the biggest eigenvalues of the covariance matrix. This reduces the number of input neurons.

The hidden neurons contain information derived from the input neurons, which is needed to build the desired input/output mapping. If the input neurons are provided with this knowledge, the network will be relieved of extracting it and representing it internally in its hidden neurons. Unfortunately, it is usually impossible to find out, «what is hidden in the hidden neurons». Therefore, one can only hope to supply the net with the «right» additional information derived from the time-series. The knowledge that economic specialists use for their prognoses is a candidate for the additional input information. Examples are

• difference \( y(t) - y(t - l) \)
  If only differences with a constant lag / are used instead of the original time series, a linear trend will be eliminated.

• up/down sign\( (y(t) - y(t - l)) \)

• maximum \( \max(y(t - l), ..., y(t - k)) \) and minimum \( \min(y(t - l), ..., y(t - k)) \) of a window

\[
\text{moving average} \quad \frac{\sum_{k=0}^{l} e^{-k} \cdot y(t-k)}{\sum_{k=0}^{l} e^{-k}}
\]

• moving standard deviation, variance, and higher moments

• trend
  • length of a trend \( \max(kly(t - k) > y(t - k + 1) > \ldots > y(t)) \), \( \max(kly(t - k) < y(t - k + 1) < \ldots < y(t)) \)
  • a combination of these transformations.

Note that adding these pretransformed time-series increases the number of input neurons if the original time-series are not omitted. But this redundancy allows for reducing the number of hidden neurons. As the hidden layers are connected to input and output layers (and other hidden layers, if available) the overall number of weights can be reduced.
• **Smoothing**

Smoothing data, such as using moving averages instead of the raw data, may eliminate noise. It is particularly interesting for continuous net outputs.

• **Reduction of training time**

If the presentation of additional knowledge to the neural net relieves it of finding it itself, this may accelerate convergence.

For other aspects of knowledge representation and coding which require preprocessing, see e.g. [Rum86]. They are not dealt with here, because they are of minor importance for our problem concerning economic time-series forecasting with backpropagation networks.

### 6.3. Network Topology

Different levels of granularity should be distinguished:

- The *global topology* on the level of networks defines how different networks are combined to a global network. Using neural nets as building blocks for new neural networks is a very promising feature which unfortunately is not comfortably supported by many neural network simulators.

- The *topology on the layer level* determines the number of clusters and their connections. For example, if a network consists of one input layer, one output layer, and two hidden layers, the following topologies are possible among others:

![Figure 5](image)

- The *topology on the neuron level* determines the number of neurons (of a layer) and their connections. These parameters are sometimes already defined on the layer level if the layers are fully connected. We will not deal with the global topology level. The layer topology is
usually defined statically during the design process and is not changed later on. Different topologies are selected for experimentation until a satisfactory solution is found. The process of finding an optimal topology can be automized by genetic algorithms [Dod91], which are very powerful if properly designed but consume much computation time and space. We obtained the best results for economic applications with very simple topologies with only one hidden layer or two parallel hidden layers with different activation functions. More complex topologies only lead to better adaptation of the training data but to poor generalization.

Modifying the topology at the node level was one of our main tools used to solve the problem of overfitting. There are two alternative strategies:

- **Resource-Allocating** [Pla91]^{12}

  Define a neural network of small size. Allocate new neurons whenever an unusual pattern is presented to the network.

- **Pruning**

  Define a neural net of sufficient size. There are rules of thumb for determining the number of hidden neurons needed [NW91]. Our recommendation concerning the incorporation of all available information, including various input data transformations, tends to lead to oversized nets.

  Prune the neural net until its generalization performance starts to decrease. Both, neuron pruning and weight pruning can be used. Input and hidden neurons can be pruned.

6.4. **Pruning**

Those neurons and weights that only serve to learn the noise instead of the structure of the data are to be eliminated. Unfortunately, there are no simple criteria for the decision, which neurons and weights ought to be pruned.

As far as input neuron pruning is concerned, the following methods can be used:

- An input neuron can be pruned if it is highly correlated with another neuron (see section 6.2).

^{12} The Resource-Allocating Network proposed by Platt for the prediction of chaotic time-series uses gaussian-like radial-basis functions as activation function. It does not apply backpropagation as learning algorithm for newly allocated units.
• The principal component analysis can be used to reduce the number of input neurons (see section 6.2).

• If the weights of all outgoing connections of a neuron are near to zero, the neuron has little influence. This happens seldom.

• A sensitivity analysis shows the influence of input neurons on the output. The neurons which are always negligible can be pruned. Two methods of sensitivity analysis are common practice:
  • Calculate the partial derivatives of the output variables with respect to the input variables. Since it is very unlikely that a derivative is almost zero for all training patterns, usually no pruning will take place. Pruning on the basis of derivatives averaged over all patterns implies the danger that special patterns, which rarely occur, are neglected.
  • Dithering [NW91]: Add a uniformly distributed noise of predefined size to each input neuron in turn and measure its influence on the output for all patterns. If the resulting variance of all output neurons is small, the input neuron can be pruned. Again, this happens very seldom.

In my experience these two methods are appropriate for analysis purposes but less suited for pruning.

Even if the output neurons are insensitive to variations of an input neuron, one cannot conclude that the information of the input neuron is irrelevant for the prediction. Other input neurons may have taken over its task.

For hidden neuron pruning similar methods can be applied:

• If two neurons of the same layer are highly correlated, they can be merged, i.e. the weights of the pruned neuron are added to the corresponding weights of the other neuron or subtracted from them if the neurons are anticorrelated.

• The principal component analysis defines new linear combinations of the neurons of a hidden layer. All incoming and outgoing weights have to be recalculated according to transformation matrix. For nonlinear dependency no such criterion is available.

• If the weights of all outgoing connections of a neuron are near to zero, the neuron can be pruned.

Weight pruning can be considered as fine grain pruning while neuron pruning is coarse grained because all incoming and outgoing weights are automatically pruned, too. Several test values for weight pruning have been proposed:

• small absolute size of the weights: $|w_{ij}|$
For example, prune 5% of all weights, or prune all weights whose absolute size is less than 10% of the average size.

- great variance of the weights: \( \sum_p (\Delta w_{ji}^p)^2 \) relative to the weight size \( w_{ji} \), i.e. weights with small test value \( \frac{\sum_p w_{ji}}{\sqrt{\sum_p (\Delta w_{ji}^p)^2}} \) are pruned\(^{13}\).

In stable networks \( \Delta w_{ji}^p \), is 0 for a perfectly learned pattern \( p \). If the presented patterns \( p \) contain noise, \( (\Delta w_{ji}^p)^2 \) will increase and the test value will decrease [FZ91].

The second criterion seems to be better suited to distinguish noise from structure.

After pruning a neural net, learning has to continue, or the weights are reinitialized and learning starts anew. It happens that weights have been pruned which otherwise would have become important at a later point of the training process. Finnoff and Zimmermann [FZ91] have developed an interesting method to undo wrong pruning decisions. The weights are pruned only temporarily, i.e. they are not used for data propagation any more, but the test values for pruning are calculated further. If they become bigger than the test values of alive weights, they can be reactivated.

Pruning is an iterative process. It is performed during the learning phase whenever the network performance starts to deteriorate.

Therefore, the network performance has to be evaluated during the training process. To this end the training data are split into a training set proper and a crossvalidation set. Only the training set proper is used for learning. The crossvalidation set is used to measure the generalization performance of the network. As a rule of thumb between 10 and 30 percent of the training data are taken for crossvalidation. Since the crossvalidation data are used indirectly to optimize the network, they should not be used to estimate the generalization performance. A generalization set that is never used during the training process serves this purpose.

\(^{13}\) The test variable becomes more complicated if training is stopped before a local minimum of the error function is reached.
6.5. Stopped Training

Another training strategy is based on the assumption that the structural relations in the training data dominate the noise and are learned first. Afterwards the noise in the data will be learned if the network is oversized. (Of course, learning of structure and of noise usually overlap, especially if the data are very noisy.) In order to gain maximum generalization performance, the training is stopped as soon as overfitting starts.

Stopped training prevents the weights from becoming very large. This reduces the net complexity if the weight size is considered as a complexity measure. Small weights usually lead to small net input of the neurons and the linear range of the activation function is used. Therefore, stopped training tends to emphasize the linear relations between input and output data.

6.6. Penalty Terms

The training strategies «pruning» and «stopped training» have a serious drawback. The number of training data is reduced by the crossvalidation set, which must not be too small in order to be representative. Both strategies aim at reducing the network complexity. This goal can also be reached by penalizing complexity through adding a penalty term to the global error function:

$$\gamma \cdot \frac{1}{2} \sum_{j,i} w_{ji}^2$$  \hspace{1cm} Weight decay [Rum86] penalizes large weights

$$\gamma \cdot \frac{1}{2} \sum_{j,i} \frac{w_{ji}^2}{1 + w_{ji}^2}$$  \hspace{1cm} Weigend/Rumelhart penalty term [HKP91] penalizes large weights without unduly penalizing very large weights.

Finnoff penalty term [Fin91] penalizes complexity measured by the metric entropy.

Any preknowledge about the input-output relation can be incorporated smoothly into the network by penalizing the deviation from the desired relation. Generally speaking, penalty functions add a bias to the predicted value but reduce its variance [Moo92].

A comparison of the three methods «stopped training», «penalty functions», and «pruning» [HFZ92] [FHZ93] shows that each has its advantages and drawbacks.
7. Results

The methods proposed in the previous paragraph were used to forecast the Deutscher Aktienindex DAX one week in advance. The main problem was the availability of relevant data. Although we had long-range values for the DAX (1980-1992), other time series were only available for the period of 1986-1992. Interesting data like daily values or maximum and minimum value during the last week could not be used.

The time-series were split as follows:

<table>
<thead>
<tr>
<th>Training data including 20% crossvalidation data</th>
<th>General</th>
</tr>
</thead>
</table>

To avoid overfitting, stopped training and weight pruning were applied. Both methods produced the same results as far as the trend prognosis is concerned.

One backpropagation network Net11DAX with 42 input neurons, 15 hidden neurons, and 1 output neuron was designed to forecast the DAX for the next week. The input values were derived from 11 time-series using windows, lags, and different transformations. Among the input time series were DAX, FAZ-Index, FAZ-Frühindikator, oil price, $US exchange rate and several indicators of customer expectations, such as put/call volume and short sales.

Another net Net11 u/d with one binary output neuron was used to predict if the DAX would increase or decrease. The results were compared with those obtained from the first net through a postprocessing step: if the forecast DAX-value was bigger than the current value, an upward trend was assumed.

Two further neural networks Net1 DAX and Net1 u/d only used a 5-weeks window of the DAX as input and had to forecast the DAX for the next week and its trend, respectively.

Moreover, the last 5 DAX-values were approximated by a straight line and by an exponential function using the least square method. These methods were also applied to the DAX time-series for the years 1986 to 1991. As expected, the correctness of their prognosis lay between 40% and 54% with a mean value of 46% for the straight line and between 40% and 56% with a mean of 50% for the exponential approximation. This prognosis is not significantly different from a random choice.
All results for the year 1992 are shown in the following table:

**Figure 9.**

<table>
<thead>
<tr>
<th>DAX Prognose</th>
<th>Net11DAX</th>
<th>Net11u/d</th>
<th>Net1DAX</th>
<th>Net1u/d</th>
<th>Line</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>% correct</td>
<td>61%</td>
<td>61%</td>
<td>57%</td>
<td>57%</td>
<td>52%</td>
<td>52%</td>
</tr>
</tbody>
</table>

The results have to be interpreted with care. If an output value varies by a small amount, an upward trend may be changed to a downward trend or vice versa. This alters the correctness by about 2%. Indeed, reinitializing Net11DAX or Net11u/d and training it again yields results between 59% and 63%. In other words, the variance is rather big and one cannot expect to obtain the same figures for all years. When forecasting the DAX for the other years using the previous years for training, the results were slightly worse because of the smaller training set, but always considerably better than the (almost random) results obtained from the regression line or the exponential approximation.

Slightly better results, namely 61%-63%, were obtained by Zimmermann et. al. [Zim 91] for the prediction of the exchange rate between the $US and the DM for 1, 2, and 3 days in advance. They used a more sophisticated neural network simulator [Her92] which contains all the features mentioned in paragraph 6.

8. **Conclusions**

Neural networks have reached a level of maturity which makes them suitable for real world applications, such as forecasting economic time-series. They are particularly interesting if only very little is known about the functional relations between input and predicted data, because they can derive the relations from training examples. The availability of a sufficiently large training set can be prohibitive, especially if part of it has to be taken for crossvalidation\(^4\). Different methods can be used to tackle the problem of generalization: stopped training, neuron and weight pruning, and penalty terms. The backpropagation network, which we used for regression (forecasting the DAX) and classification (predicting the trend) requires long

\(^4\) Multiple crossvalidation with several nets, which use different subsets of the training set for training and crossvalidation, can be used to reduce the variance of the predicted results, but it is not clear how to combine it with pruning.
training times. Combined with crossvalidation and pruning still more resources are required. The results we obtained for the short term DAX prediction are significantly better than those of simple linear regression.

REFERENCES


