Time Series Forecasting Using Neural Networks

Thomas Kolarik and Gottfried Rudorfer
Department of Applied Computer Science
Vienna University of Economics and Business Administration
Augasse 2–6, A-1090 Vienna, Austria
kolarik@wu-wien.ac.at, rudorfer@wu-wien.ac.at

Abstract

Artificial neural networks are suitable for many tasks in pattern recognition and machine learning. In this paper we present an APL system for forecasting univariate time series with artificial neural networks. Unlike conventional techniques for time series analysis, an artificial neural network needs little information about the time series data and can be applied to a broad range of problems. However, the problem of network "tuning" remains: parameters of the backpropagation algorithm as well as the network topology need to be adjusted for optimal performance. For our application, we conducted experiments to find the right parameters for a forecasting network. The artificial neural networks that were found delivered a better forecasting performance than results obtained by the well known ARIMA technique.

Motivation

Time series analysis as described by most textbooks [Cha91] relies on explicit descriptive, stochastic, spectral or other models of processes that describe the real world phenomena generating the observed data.

Usually, the parameters of a standard model like the ARIMA technique [BJ76] are derived from the autocorrelation and frequency spectrum of the time series. Problems with the ARIMA approach arise with time series of increasing variance or when the time series represents nonlinear processes.

The usage of artificial neural networks for time series analysis relies purely on the data that were observed. As multi layer feed forward networks with at least one hidden layer and a sufficient number of hidden units are capable of approximating any measurable function [HSW89, SS91], an artificial neural network is powerful enough to represent any form of time series. The capability to generalize allows artificial neural networks to learn even in the case of noisy and/or missing data. Another advantage over linear models like the ARIMA technique is the network's ability to represent nonlinear time series.

The APL programming language is very suitable for the task of implementing neural networks [Alf91, Pee81, ES91, SS93] because of its ability of handle matrix and vector operations. The forward and backward paths of a fully connected feed forward network can be implemented by outer and inner products of vectors and matrices in a few lines of APL code.

For our application, we decided to use a fully connected, layered, feed forward artificial neural network with one hidden layer and the backpropagation learning algorithm. The next section gives a short overview of the relevant definitions and algorithms.

APL 94 - 9/94 Antwerp Belgium
© 1994 ACM 0-89791-675-1/94/0009..$3.50

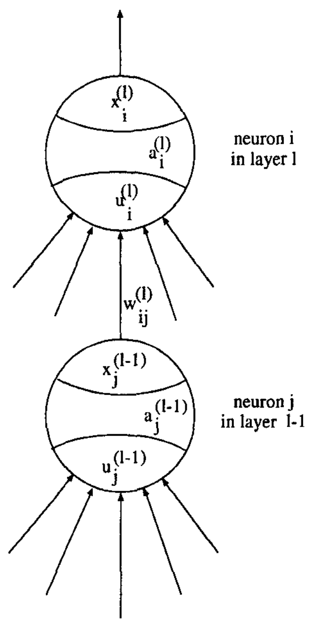


Figure 1: Exchange of activation values between neurons

A Very Short Introduction to Artificial Neural Networks

As mentioned above, our simulations utilized the "multi layered perceptron model" (MLP), also known as "feed forward networks" trained with the "generalized delta rule", also known as "backpropagation".

The foundations of the backpropagation method for learning in neural networks were laid by [RHW86].

Artificial neural networks consist of many simple processing devices (called processing elements or neurons) grouped in layers. Each layer is identified by the index $l = 0, \ldots, L$. The layers 0 and L are called the "input layer" and "output layer", all other layers are called "hidden layers". The processing elements are interconnected as follows: Communication between processing elements is only allowed for processing elements of neighbouring layers. Neurons within a layer cannot communicate. Each neuron has a certain activation level a. The network processes data by the exchange of activation levels between connected neurons (see figure 1):

The output value of the i-th neuron in layer l is denoted by $x_i^{(l)}$. It is calculated with the formula

$$x_i^{(l)} = g(a_i^{(l)})$$

where $g(\cdot)$ is a monotone increasing function. For our examples, we use the function $g(y) = \frac{1}{1+e^{-y}}$ (the "squashing function"). The activation level $a_i^{(l)}$ of the neuron i in layer l is calculated by

$$a_i^{(l)} = f(u_i^{(l)})$$

where $f(\cdot)$ is the activation function (in our case the identity function is used).

The net input $u_i^{(l)}$ of neuron i in layer l is calculated as

$$u_i^{(l)} = \left(\sum_{j=1}^{n^{(l)}} w_{ij}^{(l)} x_j^{(l-1)}\right) - \Theta_i^{(l)}$$

where $w_{ij}^{(l)}$ is the weight of neuron j in layer $l-1$ connected to neuron i in layer l, $x_j^{(l-1)}$ is the output of neuron j in layer $l-1$. $\Theta_i^{(l)}$ is a bias value that is subtracted from the sum of the weighted activations.

The calculation of the network status starts at the input layer and ends at the output layer. The input vector I initializes the activation levels of the neurons in the input layer:

$$a_i^{(0)} = \text{i}^{\text{th}} \text{ element of I}$$

For the input layer, $g(\cdot)$ is the identity function. The activation level of one layer is propagated to the next layer of the network. Then the weights between the neurons are changed by the backpropagation learning rule. The artificial neural network learns the input/output mapping by a stepwise change of the weights and minimizes the difference between the actual and desired output vector.

The simulation can be divided into two main phases during network training: A randomly selected input/output pair is presented to the input layer of the network. The activation is then propagated to the hidden layers and finally to the output layer of the network.

In the next step the actual output vector is compared with the desired result. Error values are assigned to each neuron in the output layer. The error values are propagated back from the output layer to the hidden layers. The weights are changed so that there is a lower error for a new presentation of the same pattern. The so called "generalized delta rule" is used as learning procedure in multi layered perceptron networks.

The weight change in layer l at time v is calculated by

$$\Delta w_{ij}^{(l)}(v) = \eta \delta_i^{(l)} x_j^{(l-1)} + \alpha \Delta w_{ij}^{(l)}(v-1)$$

where $\eta \in (0,1)$ is the learning rate and $\alpha \in (0,1)$ is the momentum. Both are kept constant during learning. $\delta_i^{(l)}$ is defined

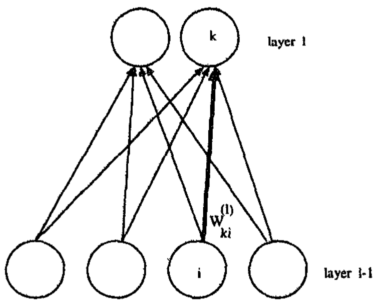


Figure 2: Weight adaptation between two neurons

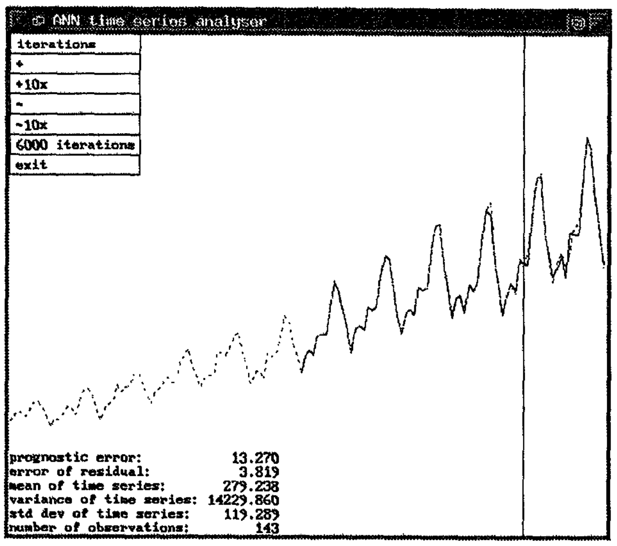
1. for the output layer (l = L) $\delta_i^{(L)}$ as
$$\delta_i^{(L)} = (d_i - x_i^{(L)})g'(u_i^{(L)})$$
where $g'(u_i^{(L)})$ is the gradient of the output function at $u_i^{(L)}$.The gradient of the output function is always positive.
The formula can be explained as follows: When the output $x_k^{(l)}$ of the neuron i in layer l is too small, $\delta_k^{(l)}$ has a negative value. Hence the output of the neuron can be raised by increasing the net input $u_k^{(l)}$ by the following change of the weight values:
if $x_i^{(l-1)} > 0$, then increase $w_{ki}^{(l)}$
if $x_i^{(l-1)} < 0$, then decrease $w_{ki}^{(l)}$
The rule applies vice versa for a neuron with an output value that is too high (see figure 2).
2. for all neurons underneath the output layer ($l < L$) $\delta_i^{(l)}$ is defined by:
$$\delta_i^{(l)} = g'(u_i^{(l)}) \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} w_{ki}^{(l+1)}$$

Finally the weights of layer l are adjusted by
$$w_{ij}^{(l,\text{new})} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)}(v)$$

Implementation

The time series modeling and forecasting system was implemented using Dyalog APL on HP 9000/700 workstations [Dya91] using the X11 window interface routines provided by the Xfns auxiliary processor. The system consists of two main components:

- A toolkit of APL functions that drive the neural network and log parameters and results of the simulation runs to APL component files.
- An X11-based graphical user interface that allows the user to navigate through the simulations and to compare the actual time series with the one generated by the neural network.



```
prognostic error:            13.270
error of residual:            3.819
mean of time series:        279.238
variance of time series:  14229.860
std dev of time series:     119.289
number of observations:         143
```

Figure 3: User interface of the forecasting system

Figure 3 presents a screen dump[1] of the user interface: the broken line shows the actual time series, the solid line represents the network's output. The forecast data is separated from the historical data – the network's training set – by the vertical bar in the right quarter of the graph. The menu in the upper left corner of figure 3 allows the user to select a view of the network's forecasting capability at different states throughout the learning phase.

By browsing through the logfiles of the simulation runs, past and present results can be compared and analyzed.

Modeling

The Training Sets

As test bed for our forecasting system we used two well known time series from [BJ76]: The monthly totals of international airline passengers (thousand of passengers) from 1949 to 1960 (see figure 4), and the daily closing prices of IBM common stock from May 1961 to November 1962 (see figure 5).

Table 1 gives some characteristics of these two time series: σ is the standard deviation, μ the mean, and n the number of observations. The airline time series is an example of time series data with a clear trend

[1]The figures for variance and mean value do not correspond with the figures in table 1 because the time series was transformed using the ∇ operator.

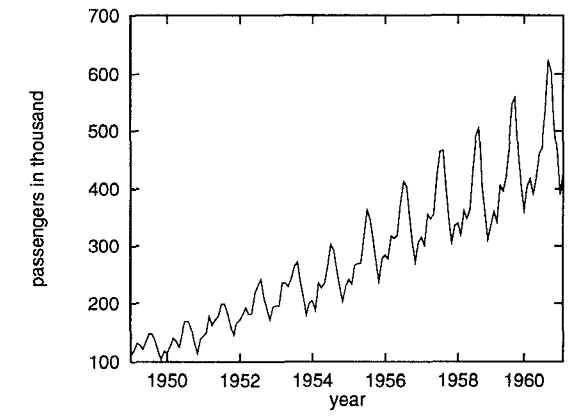


Figure 4: International airline passengers

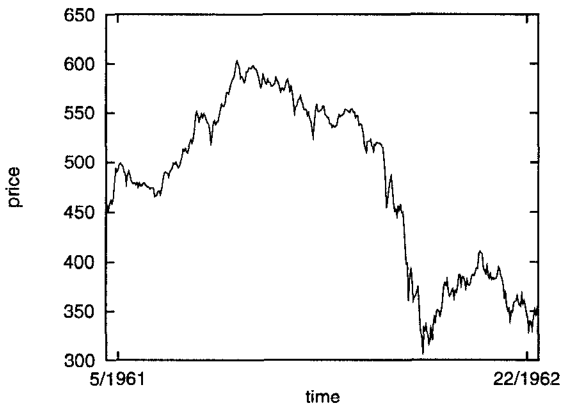


Figure 5: IBM share price

and multiplicative seasonality, whereas the IBM share price shows a break in the last third of the series and no obvious trend and/or seasonality.

Time series	σ	μ	n
IBM share price	84.11	478.47	369
airline passengers	119.55	280.30	144

Table 1: Properties of time series

The next section is concerned with the question: How can a neural network learn a time series?

The Algorithm

The neural network sees the time series $X_1, \ldots, X_n$ in the form of many mappings of an input vector to an output value (see figure 6). This technique was presented by [CMMR92].

A number of adjoining data points of the time series (the input window $X_{t-s}, X_{t-s+1} \ldots, X_t$) are mapped to the interval [0,1] and used as activation levels for the units of the input layer. The size s of the input window corresponds to the number of input units of the neural network. In a forward path, these activation levels are propagated over one hidden layer to one output unit. The error used for the backpropagation learning algorithm is now computed by comparing the value of the output unit with the transformed value of the time series at time $t + 1$. This error is propagated back to the connections between output and hidden layer and to those between hidden and output layer. After all weights have been updated accordingly, one *presentation* has been completed. Training a neural network with the backpropagation algorithm usually requires that all representations of the input set (called one *epoch*) are presented many times. In our examples, we used 60 to 138 epochs.

For the learning of time series data, the representations were presented in a randomly manner: As reported by [CMMR92], choosing a random location for each representation's input window ensures better network performance and avoids local minima.

The next section is concerned with the selection of the right parameters for the learning algorithm and the selection of a suitable topology for the forecasting network.

Network Parameters

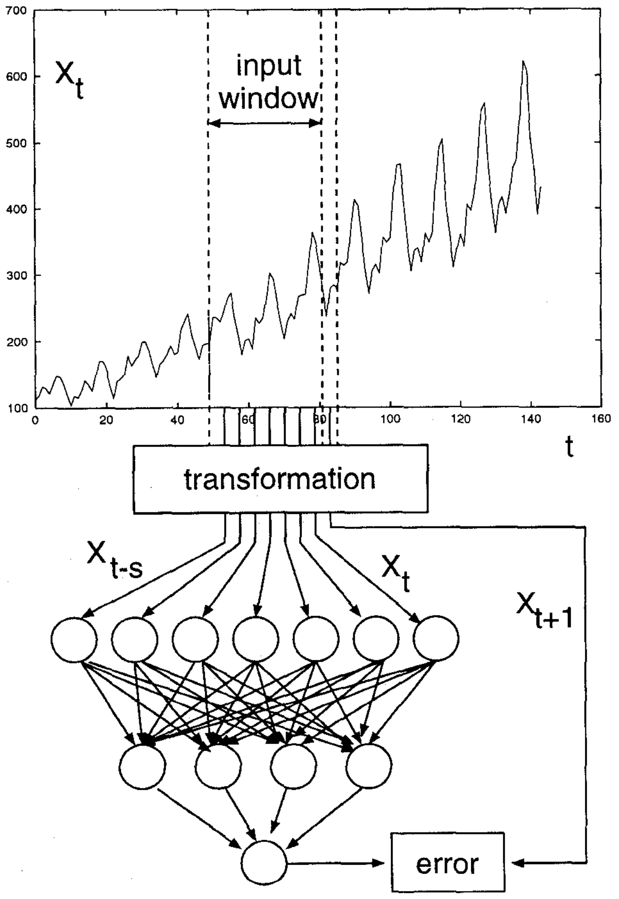


Figure 6: Learning a Time Series

The following parameters of the artificial neural network were chosen for a closer inspection:

- The learning rate η
  η $(0 < \eta < 1)$ is a scaling factor that tells the learning algorithm how strong the weights of the connections should be adjusted for a given error. A higher η can be used to speed up the learning process, but if η is too high, the algorithm will "step over" the optimum weights. The learning rate η is constant across presentations.
- The momentum α
  The momentum parameter α $(0 < \alpha < 1)$ is another number that affects the gradient descent of the weights: To prevent each connection from following every little change in the solution space immediately, the momentum term is added that keeps the direction of the previous step [HKP91], thus avoiding the descent into local minima. The momentum term is constant across presentations.
- The number of input and the number of hidden units (the network topology).
  The number of input units determines the number of periods the neural network "looks into the past" when predicting the future. The number of input units is equivalent to the size of the input window.
  Whereas it has been shown that one hidden layer is sufficient to approximate continuous function [HSW89], the number of hidden units necessary is "not known in general [HKP91]". Other approaches for time series analysis with artificial neural networks report working network topologies (number of neurons in the input-hidden-output layer) of 8-8-1, 6-6-1 [CMMR92], and 5-5-1 [Whi88].

To examine the distribution of these parameters, we conducted a number of experiments: In subsequent runs of the network, these parameters were systematically changed to explore their effect on the network's modeling and forecasting capabilities.

We used the following terms to measure the modeling quality s_m and forecasting quality s_f of our system: For a time series $X_1, \ldots, X_n$

$$s_m = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \hat{X}_i)^2}{n}} \tag{1}$$

$$s_f = \sqrt{\frac{\sum_{i=n+1}^{n+1+r} (X_i - \hat{X}_i)^2}{r}} \qquad (2)$$

where $\hat{X}_i$ is the estimate of the artificial neural network for period i and r is the number of forecasting periods. The error s_m (equation 1) estimates the capability of the neural network to mimic the known data set, the error s_f (equation 2) judges the networks's forecast capability for a forecast period of length r. In our experiments, we used $r = 20$.

Note: For reasons of clarity, in this section we only present graphics for the IBM share price time series. The graphics for the airline passenger time series are very similar.

The figures 7 and 8 demonstrate the effect of variations of the learning rate η and the momentum α on the modeling (figure 7) and forecast (figure 8) quality: both graphics give evidence for the robustness of the backpropagation algorithm, high values of both η and α should be avoided.

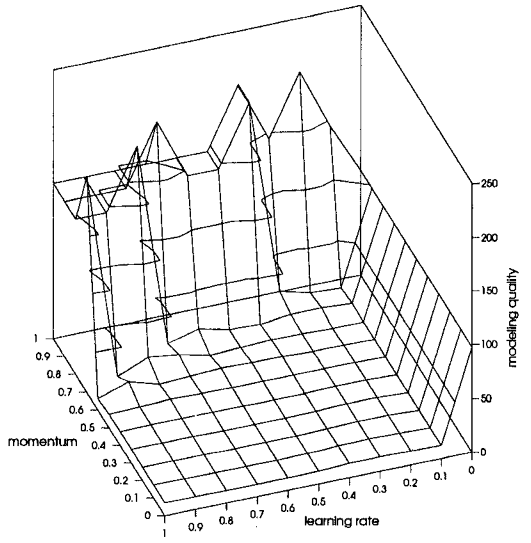


Figure 7: Learning rate and momentum, IBM share price, modeling quality

The figures 10 and 11 present the effect of different network topologies on the modeling (figure 10) and forecasting (figure 11) quality: The number of input units and the number of hidden units open an interesting view: artificial neural networks with more than approx. 50 hidden units are not suited for the task of time series forecasting. This tendency of "over-elaborate networks capable of data-miming" is also reported by [Whi88].

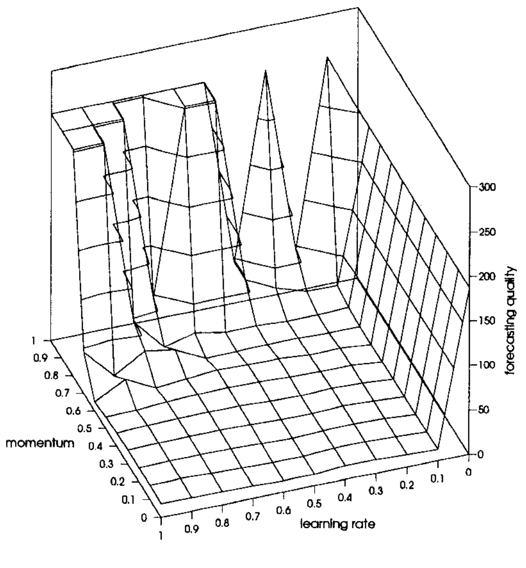


Figure 8: Learning rate and momentum, IBM share price, forecasting quality

Another parameter we have to consider is the number of presentations. A longer training period does not necessarily result in a better forecasting capability. Figure 9 demonstrates this "overlearning" effect for the IBM share price time series: with an increasing number of presentations, the network memorizes details of the time series data instead of learning its essential features. This loss of generalization power has a negative effect on the network's forecasting ability.

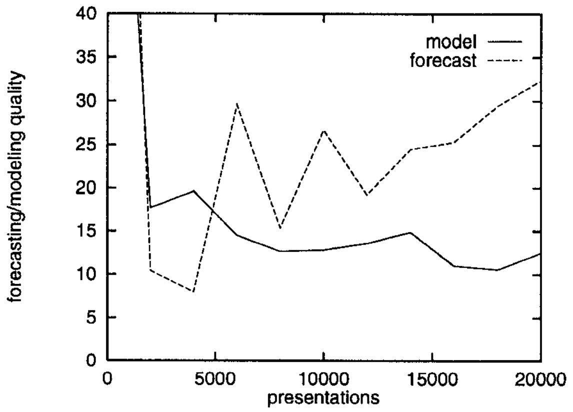


Figure 9: Modeling vs. forecasting ability

These estimations of the network's most important parameters, although rough, allowed us to choose reasonable parameters for our performance comparison with the ARIMA technique, described in the next section.

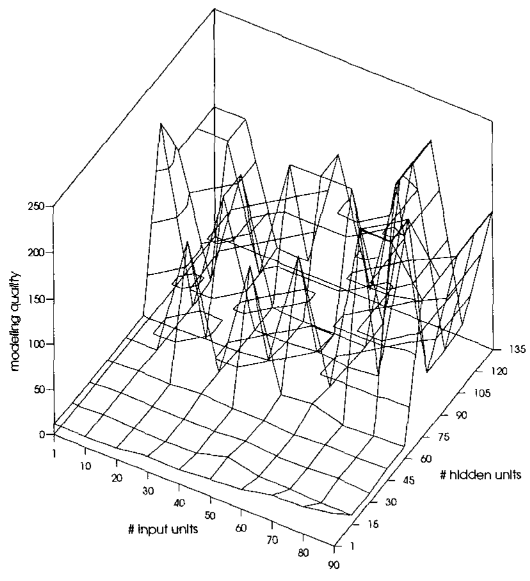


Figure 10: Number of input and hidden units, IBM share price, modeling quality

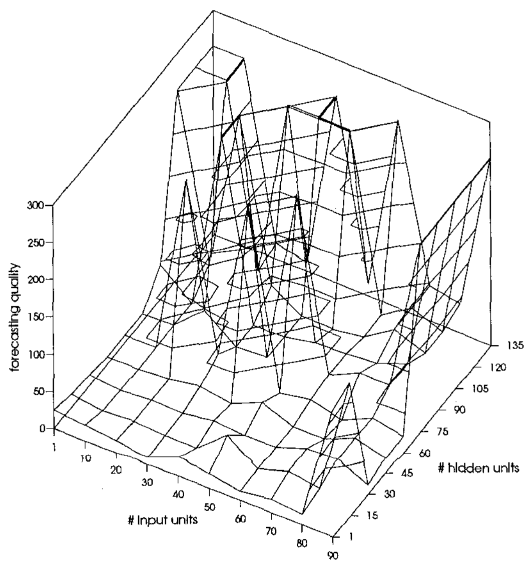


Figure 11: Number of input and hidden units, IBM share price, forecasting quality

Comparison with ARIMA Modeling

We compared our results with the results of the ARIMA procedure of the SAS software, an integrated system for data access, management, analysis and presentation. The implementation of the ARIMA procedure of SAS follows the programs described by Box and Jenkins in Part V of their classic [BJ76].

The ARIMA model is called an autoregressive integrated moving average process of order (p, d, q). It is described by the equation

$$a(z)\nabla^d X_t = b(z)U_t$$

where X_t stands for in time ordered values of a time series, $t = 1, \ldots, n$ for n observations. U_t is a sequence of random values called "white noise" process. The backward difference operator ∇ is defined as

$$\nabla X_t = X_t - X_{t-1} = (1-z)X_t$$

The variable d states how often the difference should be calculated, z is the so called backward shift operator which is defined as $z^m X_t = X_{t-m}$. The autoregressive operator $a(z)$ of order p is defined as

$$a(z) = 1 - a_1 z - a_2 z^2 - \ldots - a_p z^p$$

the moving average operator $b(z)$ of order q is defined as

$$b(z) = 1 - b_1 z - b_2 z^2 - \ldots - b_q z^q$$

We fitted an ARIMA model for each time series using the SAS system and let it predict the next 20 observations of the time series. The last 20 observations were dropped from the time series and used to calculate the prediction error of the models.

The following ARIMA models were calculated for the airline passenger time series (after a logarithmic transformation):

$$(1-z)(1-z^{12})X_t = (1 - 0.24169z - 0.47962z^{12})U_t$$

and for the IBM time series:

$$(1-z)X_t = (1 - 0.10538z)U_t$$

As an opponent for the ARIMA modeling technique, we selected those networks that delivered the smallest forecast error s_f for the respective time series data:

series	η	α	# input units	# hidden units
airline	0.1	0.9	70	45
IBM	0.1	0.9	80	30

In Table 2 the prediction errors for the artificial neural network (ANN), the artificial neural network using the logarithmic and ∇ transformation (ANN log,∇) and the ARIMA model are compared: The artificial neural network using the logarithmic and ∇ transformed time series outperformed the ARIMA models for both time series, whereas the "simple" artificial neural network predicted more accurately only for the IBM shares time series. This behavior can be explained as follows: the larger data range of the airline passenger time series leads to a loss of precision for the untransformed input set. Differencing and logarithmic transformations helped to eliminate the trend and mapped the time series data into a smaller range.

	error s_f		
Time series	ANN	ANN log,∇	ARIMA
airline passengers	20.97	13.27	18.72
IBM share price	7.97	7.70	11.35

Table 2: Forecasting errors for ANN and ARIMA model

Dyalog APL ANN Code

These functions present a complete implementation of the definitions given in the introduction. The main function NET learns the well known XOR (exclusive or) mapping. For reasons of simplicity, the input-output mapping of the XOR function is coded into the function NET.

BACKWARD

```
    ∇ Z←BACKWARD ARG;W;INPUT;G;DE;ETA;ALPHA;
      ∆WO;X;U;N;D;∆W;E
[1]    W INPUT G DE ETA ALPHA ∆WO X U←ARG
[2]    D←(E←DE-(1↓(N+1)⊃X))×((N←⍴W)⊃U)
      GRADIENT G
[3]    ∆W←⊂(ETA×(D∘.×N⊃X))+ALPHA×N⊃∆WO
[4]   MAIN:⍎(1>N←N-1)/'Z←(⌽∆W)E ⋄ →0'
[5]    D←((N⊃U)GRADIENT G)×1↓D+.×(N+1)⊃W
[6]    ∆W←∆W,⊂(ETA×(D∘.×N⊃X))+ALPHA×N⊃
      ∆WO
[7]    →MAIN
    ∇
```

FORWARD

```
    ∇ Z←FORWARD ARG;W;INPUT;G;N;X;A;U
[1]    W INPUT G←ARG
[2]    U N X←(0⍴0)(0)(1⍴⊂1,INPUT)
[3]   MAIN:⍎((⍴W)<N←N+1)/'Z←X U ⋄ →0'
[4]    U←U,⊂A←(N⊃W)+.×N⊃X
[5]    X←X,⊂1,⍎G,' A'
[6]    →MAIN
    ∇
```

GRADIENT

```
    ∇ Z←A GRADIENT G;DX;DY
[1]    DX←0.00000001
[2]    DY←(⍎G,' A+DX')-(⍎G,' A')
[3]    Z←DY÷DX
    ∇
```

NET

```
    ∇ NET;W;X;IN;G;∆W;∆WO;DE;ETA;ALPHA;U;N;Z;OS;
      IS;E;S;T;V;I;M
[1]    ETA ALPHA I M G←0.3 0.8 5000 500 'SQUASH'
[2]    IS OS←((0 0)(0 1)(1 0)(1 1))((1⍴0)(1⍴1)(
      1⍴1)(1⍴0))
[3]    S←T←V←(⍴IS)⍴0
[4]    W←(2 3⍴¯1+(6?6)÷3)(1 3⍴¯1+(3?3)÷1.5)
[5]    ∆WO N←(W×0)0
[6]   MAIN:→(I<N←N+1)/0
[7]    IN DE←(Z⊃IS)((Z←?4)⊃OS)
[8]    X U←FORWARD W IN G
[9]    ∆W E←BACKWARD W I G DE ETA ALPHA ∆WO X U
[10]   W←W+∆WO←∆W
[11]   S[Z]←S[Z]+1 ⋄ V[Z]←V[Z]+1 ⋄ T[Z]←T[Z]
      +E*2
[12]   ⍎(0=M|N)/'S,(T÷V)*÷2 ⋄ V←T←T×0'
[13]   →MAIN
    ∇
```

SQUASH

```
    ∇ Z←SQUASH A
[1]    Z←1÷1+*-A
    ∇
```

Conclusion and Further Work

We have presented a forecasting system for univariate time series that uses artificial neural networks. This computing devices proved themselves to be viable alternatives to conventional techniques. The system can be used in conjunction with other techniques for time series analysis or as a stand-alone tool.

Further work will include the comparison with other time series analysis techniques, development of hybrid techniques that combine the strength of conventional approaches with artificial neural networks and the application of our system to multivariate time series.

References

[Alf91] M. Alfonseca. Advanced applications of APL: logic programming, neural networks and hypertext. *IBM Systems Journal*, 30(4):543–553, 1991.

[BJ76] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis – forecasting and control.* Series in Time Series Analysis. Holden-Day, 500 Sansome Street, San Franciso, California, 1976.

[Cha91] E. Chatfield. *The Analysis of Time Series.* Chapman and Hall, New York, fourth edition, 1991.

[CMMR92] Kanad Chakraborty, Kishan Mehrota, Chilukuri K. Mohan, and Sanjay Ranka. Forecasting the Behaviour of Multivariate Time Series Using Neural Networks. *Neural Networks*, 5:961–970, 1992.

[Dya91] Dyadic Systems Limited, Riverside View, Basing Road, Old Basing, Basingstoke, Hampshire RG24 0AL, England. *Dyalog Apl Users Guide*, 1991.

[ES91] Richard M. Evans and Alvin J. Surkan. Relating Numbers of Processing Elements in a Sparse Distributed Memory Model to Learning Rate and Generalization. *ACM APL Quote Quad*, 21(4):166–173, 1991.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory od Neural Computation.* Addison Wesley, Redwood City, California, 1991.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

[Pee81] Howard A. Peele. Teaching A Topic in Cybernetics with APL: An Introduction to Neural Net Modelling. *ACM APL Quote Quad*, 12(1):235–239, 1981.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, October 1986.

[SS91] Hava Siegelmann and Eduardo D. Sontag. Neural Nets Are Universal Computing Devices. Technical Report SYSCON-91-08, Rutgers Center for Systems and Control, May 1991.

[SS93] Alexei N. Skurihin and Alvin J. Surkan. Identification of Parallelism in Neural Networks by Simulation with Language J. *ACM APL Quote Quad*, 24(1):230–237, 1993.

[Whi88] Halbert White. Economic prediction using neural networks: the case of ibm daily stock returns. In *Proceedings of the IEEE International Conference on Neural Networks*, pages II–451–II–459, 1988.