DEPARTAMENTO DE ECONOMIA
PUC-RIO

TEXTO PARA DISCUSSÃO
N^{o}. 446

WHAT ARE THE EFFECTS OF FORECASTING LINEAR TIME SERIES WITH NEURAL NETWORKS?

MARCELO C. MEDEIROS
CARLOS E. PEDREIRA

SETEMBRO 2001

What Are The Effects of Forecasting Linear Time Series with Neural Networks? *

Marcelo C. Medeiros
Dept. of Economics, Pontifical Catholic University of Rio de Janeiro
mcm@econ.puc-rio.br

Carlos E. Pedreira
Dept. of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro.
pedreira@ele.puc-rio.br

September 19, 2001

Abstract

This paper studies the performance of neural networks estimated with Bayesian regularization to model and forecast time series where the data generating process is in fact linear. A simulation experiment is carried out to compare the forecasts made by linear autoregressive models and neural networks.

Keywords: Neural networks, Bayesian regularization, nonlinear forecasting, nonlinear time series.

JEL Classification Codes: C22

Acknowledgments: The research of both authors has been partially supported by CNPq. We wish to thank André da Cunha for useful comments and remarks. The responsibility for any errors or shortcomings in the paper remains ours.

*Address for correspondence: Marcelo C. Medeiros, Department of Economics, Pontifical Catholic University of Rio de Janeiro, Rua Marquês de São Vicente, 225/206F Rio de Janeiro, RJ, Brazil, 22453-900.

1 Introduction

The last two decades have witnessed a vast development of nonlinear time series techniques. Among the large amount of new methodologies, Feedforward Neural Networks (NN) form an important class of nonlinear models that has attracted considerable attention in the literature. The use of these models is mainly motivated by a mathematical result stating that NN models are a universal approximator to any Borel-measurable function to any given degree of accuracy; see, for example, Funahashi (1989), Cybenko (1989), Hornik, Stinchombe, and White (1989,1990), White (1990), or Gallant and White (1992). One central topic in the NN literature is how to select the variables and the number of hidden units in the model. Usually, this is done by some "rule of thumb". A vast number of models with different combinations of variables and number of hidden units are estimated and the one with the best performance according to some known criterion is chosen as the final specification. Several alternatives to this "rule of thumb" have appeared in the literature. One strategy that turned out to be quite successful in a number of applications is Bayesian regularization, proposed by MacKay (1992a,b). The fundamental idea is to find a balance between the number of parameters and goodness of fit by penalizing large models. The objective function is modified in such a way that the estimation algorithm effectively prunes the network by driving irrelevant parameter estimates to zero during the estimation process.

The goal of this paper is to verify the effects of forecasting linear time series by neural network models estimated with Bayesian regularization. A simple simulation experiment is designed to compare the forecasting performance of neural networks and linear autoregressive models.

The plan of the paper is as follows. Section 2 describes Bayesian regularization and Section 3 discusses the issues related to forecasting from neural network models. A simulation study is carried out in Section 4. Section 5 contains concluding remarks.

2 Bayesian Regularization

Consider that the target time series $\{y_t\}$ is generated by the following stochastic process

$$y_t = T(\mathbf{x}_t; \mathbf{\Theta}) + \varepsilon_t, \tag{1}$$

where $T(\mathbf{x}_t;\boldsymbol{\Theta})$ is an unknown nonlinear function of the vector of variables $\mathbf{x}_t$ defined by the parameter vector $\boldsymbol{\Theta}$, and $\{\varepsilon_t\}$ is assumed to be a sequence of independent, normally distributed random variables with zero mean and finite variance, σ^2. Assume that $\mathbf{x}_t \in \mathbb{R}^q$ is formed by lagged values of y_t.

The goal of modelling techniques based on neural networks is to approximate (1) by the following nonlinear specification

$$y_t = G(\mathbf{z}_t;\boldsymbol{\psi}) = \lambda_0 + \sum_{i=1}^{h} \lambda_i F(\boldsymbol{\omega}_i'\mathbf{z}_t - \beta_i) + u_t, \tag{2}$$

where $G(\mathbf{z}_t;\boldsymbol{\psi})$ is a nonlinear function of $\mathbf{z}_t$, defined by the parameter vector $\boldsymbol{\psi} = [\boldsymbol{\lambda}', \boldsymbol{\omega}_1', \ldots, \boldsymbol{\omega}_h', \boldsymbol{\beta}']$, where $\boldsymbol{\lambda} = [\lambda_0, \ldots, \lambda_h]'$, $\boldsymbol{\omega}_i = [\omega_{1i}, \ldots, \omega_{pi}]'$, and $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_h]'$. The vector of variables $\mathbf{z}_t \in \mathbb{R}^p$ is formed by lagged values of y_t and its elements are called *input variables*.

Usually, $\boldsymbol{\psi}$ is estimated by nonlinear least-squares

$$\hat{\boldsymbol{\psi}} = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \tilde{Q}_T(\boldsymbol{\psi}) = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \sum_{t=1}^{N} (u_t)^2 = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \sum_{t=1}^{N} (y_t - G(\mathbf{z}_t;\boldsymbol{\psi}))^2, \tag{3}$$

and the estimated residuals $\hat{u}_t = y_t - G(\mathbf{z}_t;\hat{\boldsymbol{\psi}})$ is a good approximation to the true error term ε_t in (1). In most applications a simple gradient descent algorithm (backpropagation) is used to estimate $\boldsymbol{\psi}$. However, the estimation of $\boldsymbol{\psi}$ is usually not easy (Hush 1999).

Approximating (1) by (2) poses two main problems. First, the true vector of variables $\mathbf{x}_t$ is not known in advance and the modeller has to determine which variables should be included in $\mathbf{z}_t$. The second problem is related to the selection of the number of hidden units in (2). Selecting a small number of hidden units leads to a poor approximation of the true data generating process. On the other hand, a model with a large number of hidden units may be overfitted, generating bad forecasts (poor generalization). In most neural network applications, it is customary to select the variables and the number of hidden units of the neural network approximation using some "rule of thumb". A vast number of models with different combinations of variables and number of hidden units are estimated and the one with the best performance according to some known criterion is chosen as the final specification. Several alternatives to this "rule of thumb" have appeared in the

literature. The simplest one is the so-called *early stopping*. The key idea is to split the available data into three subsets. The first subset is used to estimate the parameters. The second subset is called the validation set. The error on the validation set is monitored during the estimation process. When the network begins to overfit the data, the error on the validation set typically begins to rise. When the validation error increases for a specified number of iterations, the estimation process is discontinued, and the parameters estimated at the minimum of the validation error serve as final estimates. The test set is not used for estimation, but it is saved for comparing different models. A large number of different specifications are estimated and compared by means of the out-of-sample performance. The model with the best forecasting performance is chosen as the final specification.

Pruning is another popular technique. The objective of pruning is to find the smallest network that fits the data well and produces good forecasts. The main idea is to start with a large network and sequentially reducing its size by removing some network connections. For a general survey on pruning see Reed (1993); see also Siestma and Dow (1988), Siestma and Dow (1991), Kaashoek and van Dijk (1998), andAnders and Korn (1999).

Another successful methodology, that is the main interest of this paper, is the Bayesian regularization approach proposed by MacKay (1992a,b). The fundamental idea is to find a balance between the number of parameters and goodness of fit by penalizing large models. The objective function is modified in such a way that the estimation algorithm effectively prunes the network by driving irrelevant parameter estimates to zero during the estimation process. The parameter vector $\boldsymbol{\psi}$ is estimated as

$$\hat{\boldsymbol{\psi}} = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \tilde{Q}_N(\boldsymbol{\psi}) = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \left(\eta Q_N(\boldsymbol{\psi}) + \gamma Q_N^*(\boldsymbol{\psi}) \right), \tag{4}$$

where $Q_N(\boldsymbol{\psi}) = \sum_{t=1}^{N} \left(y_t - G(\mathbf{z}_t; \boldsymbol{\psi}) \right)^2$, $Q_N^*(\boldsymbol{\psi})$ is the *regularization* or *penalty* term, and $\eta, \gamma > 0$ are *objective function* or *regularization* parameters. The usual penalty is the sum of squared parameters

$$Q_T^*(\boldsymbol{\psi}) = \sum_{j=0}^{p} \lambda_j^2 + \sum_{i=1}^{h} \beta_i^2 + \sum_{j=1}^{p} \sum_{i=1}^{h} \omega_{ji}^2. \tag{5}$$

The forecasting ability of the neural network model depends crucially on the values of η and γ, especially in small samples. The relative size of the objective function parameters determines the emphasis for the estimation process. If $\eta << \gamma$, then the estimation algorithm will derive

errors smaller, and the network may still overfit. If $\eta >> \gamma$, estimation will emphasize parameter size reduction at the expense of network errors, thus producing a smoother function of the input variables. The main problem with implementing regularization is setting the correct values for the objective function parameters. One approach to determine the optimal objective function parameters is the Bayesian framework, where the parameters of the network are assumed to be random variables with well-specified distributions. The objective function parameters are related to the unknown variances associated with these distributions and is estimated with statistical techniques. Foresee and Hagan (1997) give a detailed discussion of the use of Bayesian regularization in combination with the Levenberg-Marquardt optimization algorithm. The main advantage of this method is that even if the neural network model is over-parametrized, the irrelevant parameter estimates are likely to be close to zero and the model behaves like a small network.

Let $\mathbf{D} = (\mathbf{y}, \mathbf{Z})$ represent the data set, where $\mathbf{y} = [y_1, \ldots, y_N]'$ and $\mathbf{Z} = [\mathbf{z}_1 \ldots \mathbf{z}_N]'$. M a particular neural network model. After the data is taken, the density function for the parameters is updated according to Bayes' rule

$$P(\boldsymbol{\psi}|\mathbf{D}, \eta, \gamma, M) = \frac{P(\mathbf{D}|\boldsymbol{\psi}, \eta, M)\, P(\boldsymbol{\psi}|\gamma, M)}{P(\mathbf{D}|\eta, \gamma, M)}, \tag{6}$$

where $P(\boldsymbol{\psi}|\gamma, M)$ is the prior density, which represents our knowledge of the parameters before any data is collected, $P(\mathbf{D}|\boldsymbol{\psi}, \eta, M)$ is the likelihood function, which is the probability of the data occurring given the parameters and $P(\mathbf{D}|\eta, \gamma, M)$ is a normalization factor, which guarantees that the total probability is 1.

If the distribution of ε_t and the prior distribution for the parameters are both Gaussian, then $P(\mathbf{D}|\boldsymbol{\psi}, \eta, M)$ and $P(\boldsymbol{\psi}|\eta, M)$ are written as

$$P(\mathbf{D}|\boldsymbol{\psi}, \eta, M) = \left(\frac{\pi}{\eta}\right)^{-\frac{N}{2}} \exp(-\eta Q_N(\boldsymbol{\psi})) \tag{7}$$

and

$$P(\boldsymbol{\psi}|\eta, M) = \left(\frac{\pi}{\gamma}\right)^{-\frac{L}{2}} \exp(-\gamma Q_N^*(\boldsymbol{\psi})), \tag{8}$$

where $L = (p+2) \times h + 1$ is the total number of parameters in the neural network model. Substituting

(8) in (7), we get

$$P\left(\boldsymbol{\psi}|\mathbf{D},\eta,\gamma,M\right)=\frac{\left(\frac{\pi}{\eta}\right)^{-\frac{N}{2}}\left(\frac{\pi}{\gamma}\right)^{-\frac{L}{2}}\exp\left[-\left(\eta Q_N(\boldsymbol{\psi})+\gamma Q_N^*(\boldsymbol{\psi})\right)\right]}{\text{Normalization Factor}}=Z(\eta,\gamma)\exp(-\tilde{Q}_N(\boldsymbol{\psi})). \quad (9)$$

In this Bayesian framework, the optimal parameters should maximize the posterior probability $P\left(\boldsymbol{\psi}|\mathbf{D},\eta,\gamma,M\right)$, which is equivalent to minimizing the regularized objective function given in (4). The regularization parameters are optimized by applying Bayes' rule

$$P\left(\eta,\gamma|\mathbf{D},M\right)=\frac{P\left(\mathbf{D}|\eta,\gamma,M\right)P\left(\eta,\gamma|M\right)}{P\left(\mathbf{D}|M\right)}. \quad (10)$$

Assuming a uniform prior density $P\left(\eta,\gamma|M\right)$ for the regularization parameters, then maximizing the posterior is achieved by maximizing the likelihood function $P\left(\mathbf{D}|\boldsymbol{\psi},\eta,M\right)$. Since all probabilities have a Gaussian form, the normalization factor is expressed as

$$P\left(\mathbf{D}|\eta,\gamma,M\right)=\frac{P\left(\mathbf{D}|\boldsymbol{\psi},\eta,M\right)P\left(\boldsymbol{\psi}|\gamma,M\right)}{P\left(\boldsymbol{\psi}|\mathbf{D},\eta,\gamma,M\right)}=\left(\frac{\pi}{\eta}\right)^{-\frac{N}{2}}\left(\frac{\pi}{\gamma}\right)^{-\frac{L}{2}}Z^{-1}(\eta,\gamma). \quad (11)$$

Since the objective function is quadratic in a small area surrounding a minimum point, we can expand $\tilde{Q}_N(\boldsymbol{\psi})$ in a Taylor series around the minimum point of the posterior density, where the gradient is zero. Solving for the normalizing constant yields

$$Z(\eta,\gamma)=(2\pi)^{\frac{L}{2}}\left[\det\left(\mathbf{H}\right)^{-1}\right]^{1/2}\exp(-\tilde{Q}_N(\boldsymbol{\psi})), \quad (12)$$

where $\mathbf{H}$ is the Hessian matrix of the objective function. Inserting (11) into (10) we solve for the optimal values for η and γ at the minimum point. We do this by taking the derivative with respect to the log of (11) at set them equal to zero. This yields

$$\hat{\gamma}=\frac{\kappa}{2Q_N^*(\boldsymbol{\psi})} \quad (13)$$

and

$$\hat{\eta}=\frac{N-\kappa}{2Q_N(\boldsymbol{\psi})}, \quad (14)$$

where $\kappa = L - 2\gamma \text{trace}(\mathbf{H})^{-1}$ is called the effective number of parameters.

Following Foresee and Hagan (1997), here are the steps required for Bayesian optimization of the regularization parameters, with the Gauss-Newton approximation to the Hessian matrix:

1. Initialize η, γ, and the network parameters by the Nguyen-Widrow rule (Nguyen and Widrow 1990). After the first estimation step, the objective function parameters will recover from the initial setting.

2. Take one step of the Levenberg-Marquardt algorithm to minimize the objective function $\tilde{Q}_N(\boldsymbol{\psi})$.

3. Compute the effective number of parameters $\kappa = L - 2\gamma \text{trace}(\mathbf{H})^{-1}$ making use of the Gauss-Newton approximation to the Hessian matrix available in the Levenberg-Marquardt optimization algorithm: $\mathbf{H} = \nabla^2 \tilde{Q}_N(\boldsymbol{\psi}) \approx 2\gamma \mathbf{J}'\mathbf{J} + 2\eta \mathbf{I}_N$, where $\mathbf{J}$ is the Jacobian matrix of the estimation set errors.

4. Compute new estimates for the objective function parameters.

5. Now iterate steps 1 through 3 until convergence.

3 Forecasting with Neural Network Models

Multi-step forecasting with nonlinear models is more challenging than forecasting with linear models. See, for example, Granger and Teräsvirta (1993, Section 8.1) for a general discussion.

Consider the simple nonlinear model defined as

$$y_t = G(y_{t-1}; \boldsymbol{\psi}) + u_t, \tag{15}$$

where $G(\cdot)$ is a nonlinear function with parameter vector $\boldsymbol{\psi}$. The term u_t is an independent identically distributed random variable with zero mean and finite variance. The history of the process up to time t is called $\mathcal{I}_t$.

Due the fact that $E(u_{t+1}|\mathcal{I}_t) = 0$, the optimal 1-step-ahead forecast of y_{t+1} is given by

$$\hat{y}_{t+1|t} = E(y_{t+1}|\mathcal{I}_t) = G(y_t; \boldsymbol{\psi}), \tag{16}$$

which is equivalent to the optimal 1-step-ahead forecast when $G(\cdot)$ is linear.

For multi-step forecasts, the problem is much more complicated. For 2-step-ahead the optimal forecast is given by

$$\hat{y}_{t+2|t} = E(y_{t+2}|\mathcal{I}_t) = E(G(y_{t+1};\boldsymbol{\psi})|\mathcal{I}_t)) = \int_{-\infty}^{\infty} G(y_{t+1};\boldsymbol{\psi}) f(u_{t+1}) du_{t+1}, \tag{17}$$

where $f(u_{t+1})$ is the density of u_{t+1}. Usually the expression (17) is approximated by numerical techniques, such as, for example, monte-carlo or bootstrap.

The monte-carlo method is a simple simulation technique for obtaining multi-step forecasts. For model (15), the k-step-ahead forecast is defined as

$$\hat{y}_{t+k|t} = \frac{1}{M} \sum_{i=1}^{M} \hat{y}_{t+k|t}^{(i)}, \tag{18}$$

where M is the number of replications and

$$\hat{y}_{t+k|t}^{(i)} = G(\hat{y}_{t+k-1|t});\boldsymbol{\psi}) + \xi_{t+k|t}^{(i)}. \tag{19}$$

$\xi_{t+k|t}^{(i)}$ is a random number drawn from a normal distribution with the same mean and standard deviation as the in-sample estimated residuals.

4 Monte Carlo Simulation

In this section we report the results of a simulation study designed to find out the behavior of neural networks estimated with Bayesian regularization when the main concern is multi-step forecasting. We simulate 500 replications of several first order autoregressive (AR) models defined by the following equation

$$y_t = \phi y_{t-1} + \varepsilon_t, \quad \varepsilon_t \sim \text{NID}(0, 0.5^2), \tag{20}$$

where $\phi = 0.65, 0.75, 0.85, 0.95$. For each replication we discard the first 500 observations to avoid any initialization effects.

The forecasting experiment can be viewed of consisting of the following steps.

1. For each monte-carlo iteration

   (a) Generate a time series by the process defined in (20) with N observations, $N = 150, 550$.

   (b) Split the sample into two subsamples: the estimation set ($t = 1, \ldots, t_0$) and the forecasting set ($t = t_0 + 1, \ldots, N$), where $t_0 = 100, 500$.

   (c) Estimate the parameters of a neural network model with 5 hidden units and the first six lags as input variables and a first order linear autoregressive model with drift using only the estimation set.

   (d) For $t = t_0, \ldots, N - 4$, compute the k-steps-ahead out-of-sample forecasts, $k = 1, \ldots, 4$, denoted by $\hat{y}_{t+k|t}$, and the associated forecast errors $\hat{u}_{t+k|t}$. Multi-step forecasts for the neural network models are obtained by monte-carlo simulation with 500 iterations.

   (e) For each forecasting horizon, compute the root mean square errors and the mean absolute errors defined as

$$RMSE(k) = \sqrt{\frac{1}{N - t_0 - 3} \sum_{t=N}^{N-4} \hat{u}_{t+k|t}^2}, \tag{21}$$

$$MAE(k) = \frac{1}{N - t_0 - 3} \sum_{t=t_0}^{N-4} |\hat{u}_{t+k|t}|. \tag{22}$$

2. Compute the mean and the standard deviation of the performance measures (21) and (22) for each forecast horizon over the 500 replications.

Tables 1 and 2 show the mean and the standard deviation (between parenthesis) of the RMSE and MAE over the 500 repetitions. The results in Table 1 concern time series with 100 observations each and the ones in Table 2 refer to 500 observations. Observing Tables 1 and 2 we can see that the linear AR(1) model produces forecasts with lower RMSE and MAE than the neural network model and that the difference increases as ϕ tends to 1. When the sample size is increased the difference between the forecasts made by the linear and the neural network models is smaller.

5 Conclusions

This paper has addressed the issue of forecasting linear time series with neural networks estimated with Bayesian regularization. Bayesian regularization is a technique designed to avoid overfitting,

Table 1: Mean and standard deviation (between parenthesis) of the root mean square errors and the mean absolute errors of multi-step forecasts over 500 replications of model (20) (100 observations).

Horizon	$\phi = 0.65$				$\phi = 0.75$			
	Neural Network		Linear model		Neural Network		Linear model	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	0.5292 (0.0694)	0.4252 (0.0590)	0.5031 (0.0533)	0.4036 (0.0449)	0.5372 (0.0792)	0.4308 (0.0647)	0.5058 (0.0558)	0.4056 (0.0465)
2	0.6275 (0.0916)	0.5059 (0.0762)	0.6004 (0.0750)	0.4822 (0.0620)	0.6731 (0.1078)	0.5426 (0.0894)	0.6352 (0.0861)	0.5124 (0.0717)
3	0.6628 (0.1071)	0.5360 (0.0891)	0.6378 (0.0894)	0.5133 (0.0732)	0.7358 (0.1260)	0.5942 (0.1067)	0.6974 (0.1056)	0.5631 (0.0886)
4	0.6762 (0.1171)	0.5478 (0.0985)	0.6541 (0.0981)	0.5276 (0.0819)	0.7662 (0.1388)	0.6210 (0.1180)	0.7302 (0.1197)	0.5915 (0.1006)

Horizon	$\phi = 0.85$				$\phi = 0.95$			
	Neural Network		Linear model		Neural Network		Linear model	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	0.5535 (0.0987)	0.4437 (0.0808)	0.5059 (0.0556)	0.4060 (0.0480)	0.6362 (0.2570)	0.5128 (0.2109)	0.5144 (0.0752)	0.4141 (0.0648)
2	0.7340 (0.1531)	0.5925 (0.1287)	0.6686 (0.0978)	0.5392 (0.0833)	0.8979 (0.3809)	0.7305 (0.3208)	0.7237 (0.1409)	0.5841 (0.1246)
3	0.8366 (0.1832)	0.6803 (0.1569)	0.7679 (0.1322)	0.6222 (0.1135)	1.0707 (0.4573)	0.8794 (0.3974)	0.8741 (0.2033)	0.7121 (0.1833)
4	0.9015 (0.2069)	0.7361 (0.1790)	0.8318 (0.1587)	0.6770 (0.1376)	1.1952 (0.5120)	0.9886 (0.4529)	0.9924 (0.2593)	0.8134 (0.2335)

Table 2: Mean and standard deviation (between parenthesis) of the root mean square errors and the mean absolute errors of multi-step forecasts over 500 replications of model (20) (500 observations).

Horizon	$\phi = 0.65$				$\phi = 0.75$			
	Neural Network		Linear model		Neural Network		Linear model	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	0.5045 (0.0554)	0.4047 (0.0448)	0.4984 (0.0533)	0.3999 (0.0434)	0.5092 (0.0561)	0.4082 (0.0467)	0.5028 (0.0519)	0.4030 (0.0434)
2	0.6028 (0.0781)	0.4854 (0.0661)	0.5952 (0.0745)	0.4792 (0.0634)	0.6324 (0.0870)	0.5091 (0.0725)	0.6237 (0.0799)	0.5019 (0.0667)
3	0.6360 (0.0934)	0.5127 (0.0787)	0.6282 (0.0899)	0.5060 (0.0759)	0.6919 (0.1084)	0.5585 (0.0915)	0.6829 (0.1012)	0.5511 (0.0854)
4	0.6470 (0.0998)	0.5211 (0.0836)	0.6406 (0.0969)	0.5157 (0.0812)	0.7217 (0.1216)	0.5855 (0.1046)	0.7129 (0.1148)	0.5775 (0.0983)
Horizon	$\phi = 0.85$				$\phi = 0.95$			
	Neural Network		Linear model		Neural Network		Linear model	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	0.5043 (0.0574)	0.4049 (0.0490)	0.4969 (0.0556)	0.3986 (0.0448)	0.5117 (0.0627)	0.4106 (0.0533)	0.5004 (0.0506)	0.4009 (0.0438)
2	0.6602 (0.0902)	0.5320 (0.0775)	0.6501 (0.0818)	0.5233 (0.0694)	0.7085 (0.1122)	0.5702 (0.0957)	0.6897 (0.0891)	0.5533 (0.0766)
3	0.7544 (0.1138)	0.6091 (0.0958)	0.7428 (0.1049)	0.5992 (0.0875)	0.8427 (0.1556)	0.6822 (0.1335)	0.8197 (0.1260)	0.6616 (0.1079)
4	0.8123 (0.1333)	0.6574 (0.1131)	0.8004 (0.1260)	0.6474 (0.1056)	0.9467 (0.1945)	0.7705 (0.1678)	0.9204 (0.1606)	0.7467 (0.1368)

where the fundamental idea is to find a balance between the number of parameters and the goodness of fit by penalizing large models. The objective function is modified in such a way that the estimation algorithm effectively prunes the network by driving irrelevant parameter estimates to zero during the estimation process. The main advantage of this method is that even if the ANN model is over-parametrized, the irrelevant parameter estimates are likely to be close to zero and the model behaves like a small network. A simulation study has been carried out to evaluate the performance of neural networks models in forecasting time series generated by a simple linear autoregressive model. The results were compared with the ones obtained by a linear autoregressive model with drift. The main conclusion is that, in small samples, neural networks produces forecasts with larger RMSE and MAE statistics than the linear models, specially the the data generating process is close to a unit-root model. When the sample size is increased the difference between linear and nonlinear forecasts becomes smaller.

References

Anders, U. and Korn, O.: 1999, Model selection in neural networks, *Neural Networks* **12**, 309–323.

Cybenko, G.: 1989, Approximation by superposition of sigmoidal functions, *Mathematics of Control, Signals, and Systems* **2**, 303–314.

Foresee, F. D. and Hagan, M. . T.: 1997, Gauss-newton approximation to Bayesian regularization, *IEEE International Conference on Neural Networks (Vol. 3)*, IEEE, New York, pp. 1930–1935.

Funahashi, K.: 1989, On the approximate realization of continuous mappings by neural networks, *Neural Networks* **2**, 183–192.

Gallant, A. R. and White, H.: 1992, On learning the derivatives of an unknown mapping with multilayer feedforward networks, *Neural Networks* **5**, 129–138.

Granger, C. W. J. and Teräsvirta, T.: 1993, *Modelling Nonlinear Economic Relationships*, Oxford University Press, Oxford.

Hornik, K., Stinchombe, M., and White, H.: 1990, Universal approximation of an unknown mapping and its derivatives using multi-layer feedforward networks, *Neural Networks* **3**, 551–560.

Hornik, K., Stinchombe, M. and White, H.: 1989, Multi-layer Feedforward networks are universal approximators, *Neural Networks* **2**, 359–366.

Hush, J.: 1999, Training a sigmoidal node is hard, *Neural Computation* **11**(5), 1249–1260.

Kaashoek, J. F. and van Dijk, H. K.: 1998, A simple strategy to prune neural networks with an application to economic time series, *Econometric Institute Report 9854/A*, Econometric Institute – Erasmus University.

MacKay, D. J. C.: 1992a, Bayesian interpolation, *Neural Computation* **4**, 415–447.

MacKay, D. J. C.: 1992b, A practical bayesian framework for backpropagation networks, *Neural Computation* **4**, 448–472.

Nguyen, D. and Widrow, B.: 1990, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, pp. 21–26.

Reed, R.: 1993, Prunning algorithms – a survey, *IEEE Transactions on Neural Networks* **4**, 740–747.

Reed, R. D. and Marks II, R. J.: 1999, *Neural Smithing*, MIT Press, Cambridge, MA.

Siestma, J. and Dow, R. J. F.: 1988, Neural net pruning – why and how, *IEEE International Conference on Neural Networks*, Vol. 1, pp. 325–333.

Siestma, J. and Dow, R. J. F.: 1991, Creating artificial neural networks that generalize, *Neural Networks* **4**(1), 67–69.

White, H.: 1990, Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings, *Neural Networks* **3**, 535–550.