Macroeconomic time series prediction using prediction networks and evolutionary algorithms

P. Forsberg & M. Wahde
Department of Applied Mechanics, Chalmers University of Technology, Sweden

Abstract

The prediction of macroeconomic time series by means of a form of fully recurrent neural networks, called discrete-time prediction networks (DTPNs), is considered. The DTPNs are generated using an evolutionary algorithm, allowing both structural and parametric modifications of the networks, as well as modifications in the squashing function of individual neurons.

The results show that the evolved DTPNs achieve better performance on both training and validation data compared to benchmark prediction methods. The importance of allowing structural modifications in the evolving networks is discussed. Finally, a brief investigation of predictability measures is presented.
Key words: time series prediction, recurrent neural networks, evolutionary algorithms.

1 Introduction

Prediction of time series is an important problem in many fields, including economics. Due to the high level of noise in macroeconomic time series, models involving two parts, one deterministic and one stochastic, are often used. One such method is ARIMA [1]. For one-step prediction, the results obtained by these simple predictive methods (such as exponential smoothing, which is a special case of ARIMA models), are difficult to improve much due to the high levels of noise present. However, even a small improvement can translate into considerable amounts of money for data sets that concern e.g. an entire national economy. The aims of this paper is (1) to introduce a class of generalized, recurrent neural networks and an associated evolutionary optimization method and (2) to apply such networks to the problem of deterministic prediction of macroeconomic time series, with the aim of extracting as much information as possible, while keeping in mind that the noise in the data introduces limits on the achievable performance.

2 Macroeconomic data

Two different data sets were considered, namely US GDP (quarterly variation, from 1947, first quarter to 2005, second quarter), and the Fed Funds interest rate (monthly values, from July 1954 to July 2005). The raw GDP and interest data were first transformed to a relative difference series, using the transformation

$$Z_{\mathrm{RD}}(t) = \frac{Z_{\mathrm{raw}}(t) - Z_{\mathrm{raw}}(t-1)}{Z_{\mathrm{raw}}(t-1)}. \tag{1}$$

Next, this series was further transformed using a hyperbolic tangent transformation

$$Z(t) = \tanh(C_{\mathrm{TH}} Z_{\mathrm{RD}}(t)). \tag{2}$$

For the GDP and interest rate series transformations, the values $C_{\mathrm{TH}} = 25$ and $C_{\mathrm{TH}} = 5$ were used, respectively. The aim of the hyperbolic tangent transformation was to make the data points as evenly distributed as possible in the range $[-1, 1]$.

Both data sets were divided into a training part with M_{tr} data points, and a validation part with M_{val} data points. During training, only the results (i.e. the error) over the training data set were used as feedback to the optimization procedure (see below). The rescaled GDP data set contained 233 data points. For training, steps 16-115 were used ($M_{\mathrm{tr}} = 100$) and for validation, steps 126-225 were used ($M_{\mathrm{val}} = 100$). During training, the first 15 steps were used to initialize the short-term memory of the DTPN. A similar initialization procedure was applied during validation. For the Fed Funds data set, with 612 data points, steps 26-475 were used for training ($M_{\mathrm{tr}} = 450$) and steps 486-605 ($M_{\mathrm{val}} = 120$) were used for validation.

3 Methods for prediction

3.1 Discrete-time prediction networks

Neural networks constitute a commonly used blackbox prediction model. In most cases, feedforward neural networks (FFNNs) are used. In such networks, the computational elements (neurons) are placed in layers. The input signals (i.e. earlier, consecutive values of the time series) are distributed to the neurons in the first layer, and the output signals of those neurons are then computed and used as input in the second layer etc. The output of a given neuron i is computed as

$$x_i(t+1) = \sigma\left(b_i + \sum_{j=1}^{N} w_{ij} y_j\right), \tag{3}$$

where b_i is the bias term, w_{ij} are the weights connecting neuron j in the preceding layer to neuron i, N is the number of neurons in the preceding layer, and σ is the squashing function, usually taken as the logistic function

$$\sigma_1(z) = \frac{1}{1+\mathrm{e}^{-cz}}, \tag{4}$$

where c is a positive constant, or the hyperbolic tangent

$$\sigma_2(z) = \tanh cz. \tag{5}$$

Given a set of training data, i.e. a list of input vectors and their corresponding desired output, such networks can be trained using gradient-based methods, such as e.g. backpropagation.

However, there are fundamental limitations in the prediction that can be achieved using FFNNs, due to their lack of dynamic (short-term) memory. Stated differently, an FFNN will, for a given input, always give the same output, regardless of any earlier input signals [2, 3]. Thus, such networks are unable to deal with situations in which identical inputs to the network (at different times along the time series) require different outputs. Earlier work [2] has shown that dynamic short-term memory *does* make a difference in neural network-based time series prediction.

Furthermore, the requirement that it should be possible to obtain a gradient of the prediction error, in order to form the derivatives needed for updating the weights (during training), restricts the shape of the squashing functions. Without such restrictions, squashing functions such as e.g.

$$\sigma_3(z) = \mathrm{sgn}(z), \tag{6}$$

and

$$\sigma_4(z) = \begin{cases} \tanh(z+c) & \text{if} \quad z < -c \\ 0 & \text{if} \quad -c \le z \le c \\ \tanh(z-c) & \text{if} \quad z > c \end{cases} \tag{7}$$

could be used.

To overcome the limitations of FFNNs, it is possible to introduce feedback couplings in the networks, transforming them into recurrent neural networks (RNNs). Such networks have been used in many financial and macroeconomic applications, see e.g. [3, 4]. A problem with many standard training techniques for neural networks is that they require that the user should set the structure of the network (i.e. the number of neurons and their position in the network), a procedure for which one often has to rely on guesswork and rules-of-thumb [5]. An alternative training procedure is to use an evolutionary algorithm (EA) which, if properly designed, can handle both structural and parametric optimization [6].

In this paper, a new kind of network (and an associated evolutionary optimization method), well suited for the problem of time series prediction, will be used, with dynamical memory, arbitrary structure, and (in principle) arbitrary squashing functions. Each of the n neurons in these networks which, henceforth, will be called *discrete-time prediction networks* or DTPNs for short) contains arbitrary connections from the n_{in} input elements and from other neurons (including itself). In addition, each neuron has an *evaluation order tag* (EOT) such that, in each time step, the output of the neurons with the lowest EOT values is computed first, followed by the output of the neurons with the second lowest EOT values etc. The output neuron, i.e. the neuron with highest EOT (arbitrarily chosen as neuron 1) is evaluated last. Thus, the equations for neurons with the lowest EOT become

$$x_i(t+1) = \sigma\left(b_i + \sum_{j=1}^{n_{\text{in}}} w_{ij}^{\text{in}} I_j(t) + \sum_{j=1}^{n} w_{ij} x_j(t)\right), \tag{8}$$

where w_{ij}^{in} are the input weights, w_{ij} the interneuron weights, and b_i is the bias term. I_j are the inputs to the network which, in the case of time series prediction, consist of earlier values of the time series $Z(t)$, i.e. $I_j(t) = Z(t-j+1)$. The number of inputs can thus be referred to as the *lookback* (L) of the DTPN. For neurons with the second lowest EOT, the equations look the same, except that $x(t)$ is changed to $x(t+1)$ for neurons with lowest EOT etc. Finally, the output neuron gives the following output

$$x_1(t+1) = \sigma\left(b_1 + \sum_{j=1}^{n_{\text{in}}} w_{1j}^{\text{in}} I_j(t) + w_{11}x_1(t) + \sum_{j=2}^{n} w_{1j} x_j(t+1)\right), \tag{9}$$

since, at this stage, all neurons except neuron 1 have been updated. It is evident that the EOTs introduce the equivalent of layers. Thus, while most DTPNs will contain many recurrent connections, an FFNN is a special case of a DTPN. More precisely, a DTPN is equivalent to an ordinary FFNN if and *only* if (1) all squashing functions are of the same type (either σ_1 or σ_2), (2) only neurons with the lowest EOT values receive external input, and (3) w_{ij} (i.e. the weight connecting neuron j to neuron i) is equal to zero if EOT$(j) \geq$ EOT(i).

3.2 Benchmark predictions

In order to evaluate the results obtained using DTPNs, a comparison will be made with two standard prediction techniques, namely *autoregressive moving average* (ARMA) and *exponential smoothing*. The general simple ARMA(p,q) model

$$\phi(\Lambda)Z(t) = \theta(\Lambda)\epsilon(t), \tag{10}$$

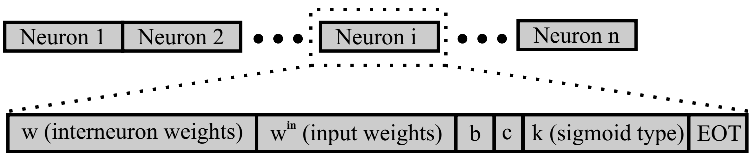

Figure 1: A chromosome encoding a DTPN.

where Λ is the lag operator, ϵ is the disturbance $Z - \hat{Z}$, and

$$\phi(\Lambda) = 1 - \phi_1\Lambda - \ldots - \phi_p\Lambda^p, \tag{11}$$

and

$$\theta(\Lambda) = 1 + \theta_1\Lambda + \ldots + \theta_q\Lambda^q, \tag{12}$$

gives the one-step prediction $\hat{Z}(t+1|t)$

$$\hat{Z}(t+1|t) = \sum_{i=0}^{p} \phi_i Z(t-i) + \sum_{i=0}^{q} \theta_i \epsilon(t-i). \tag{13}$$

ϕ_i and θ_i are parameters to be estimated in order to find the lowest error. The exponential smoothing technique (without trend) is described by the ARIMA(0,1,1) equation

$$(1-\Lambda)Z(t) = (1-\theta_1\Lambda)\epsilon(t). \tag{14}$$

This model gives the prediction

$$\hat{Z}(t+1|t) = \frac{1-\theta_1}{1-\theta_1\Lambda}Z(t) = \theta_1\hat{Z}(t|t-1) + (1-\theta_1)Z(t). \tag{15}$$

As a special case, if $\theta_1 = 0$, the *naive prediction* $\hat{Z}(t+1|t) = Z(t)$ is obtained.

4 Evolutionary algorithm

The DTPNs were generated using an evolutionary algorithm (EA) [7]. The EA used here employed a non-standard chromosomal representation, shown in Fig. 1, in which each gene represented a neuron in the network, encoding its interneuron weights (w_{ij}), input weights (w_{ij}^{in}), bias term (b_i), sigmoid parameter (c), sigmoid type, and EOT. During the formation of new individuals, crossover was only allowed between individuals containing the same number of neurons. Several different forms of mutations were used, both parametric mutations modifying the values of the parameters (including the EOT) listed above, and structural mutations which could either add or subtract a neuron from the DTPN. No upper limit was set on the number of neurons. A lower limit of 2 neurons was introduced, however. In addition to the mutations just listed, a sigmoid type mutation was introduced as well, allowing a neuron to change its sigmoid type by randomly changing the index k of the sigmoid σ_k (see Sect. 3.1 and Eq. (17) below). Finally, in order to *allow* (not force) the EA to produce sparsely connected networks, some runs were carried out in which parametric mutations of interneuron weights, input weights, and biases not only could modify the value of the parameter in question, but also (with low probability) could set it exactly to zero. Thus, these mutations essentially functioned as on-off toggles, and were therefore called *zero-toggle mutations*. The number of input elements (and therefore the lookback L) was fixed in each run. The fitness measure F used by the EA was taken as the inverse of the RMS prediction error over the training set, i.e. $F = 1/e_{\text{RMS}}$ where

$$e_{\text{RMS}} = \sqrt{\frac{1}{M_{\text{tr}}} \sum_{i=1}^{M_{\text{tr}}} \left(Z(i) - \hat{Z}(i) \right)^2} \tag{16}$$

Note that the use of an EA implies that any form of sigmoid function can be used in the networks. In addition to the four functions $\sigma_1 - \sigma_4$, a fifth sigmoid, namely

$$\sigma_5(z) = \frac{cz}{1 + (cz)^2}, \tag{17}$$

was also allowed in the simulations reported below.

5 Prediction results

A large number of runs were carried out, using different number of inputs and different EA parameters in order to test the ability of the evolutionary algorithm to generate DTPNs with low prediction error for the two data sets under consideration.

The results are summarized in Table 1. The table shows the prediction error for the DTPN with lowest validation error. In addition, the prediction errors obtained using naive prediction, exponential smoothing, and ARMA (all with optimized parameter values), are shown.

As is evident from the table, the best DTPNs outperform the two other prediction methods. Table 2 gives a more detailed description of the best DTPNs, obtained with different values of n_{in}. For comparison, note that the best *training* errors obtained with exponential smoothing were $e_{\text{ES}}^{\text{tr}} = 0.2512$ for the GDP data and $e_{\text{ES}}^{\text{tr}} = 0.3477$ for the Fed funds data. Using the ARMA model, the best training errors were $e_{\text{ARMA}}^{\text{tr}} = 0.2108$ and $e_{\text{ARMA}}^{\text{tr}} = 0.3248$, respectively.

6 Predictability measures

The fact that the DTPNs outperform the benchmark prediction methods does not imply that these networks extract all the available information in the time series under study. One way of determining whether additional information can be extracted would be to devise a measure $P(t)$ of predictability such that, in addition

Table 1: Minimum errors over the *validation* part of the data set, obtained using naive prediction (e_{N}), exponential smoothing (e_{ES}), ARMA (e_{ARMA}), and DTPNs (e_{DTPN}). Only the results for the very best DTPN are shown.

Data set	e_{N}	e_{ES}	e_{ARMA}	e_{DTPN}
Fed funds interest rate	0.2018	0.1901	0.1887	0.1837
GDP	0.1771	0.1490	0.1473	0.1305

Table 2: Examples of the performance of evolved DTPNs. The second column shows the number of inputs to the network, and the third column shows the probability of a mutation being of the zero-toggle type, i.e. a mutation that sets the parameter in question to zero. The fourth column shows the (evolved) number of neurons, and the fifth column shows the (evolved) number of layers (n_L), i.e. the number of distinct EOT values in the evolved network. The two final columns show the errors over the training and validation parts of the data set.

Data set	n_{IN}	P_{zero}	n	n_L	$e^{\mathrm{tr}}_{\mathrm{DTPN}}$	$e^{\mathrm{val}}_{\mathrm{DTPN}}$
Fed funds, run 1	2	0.00	7	5	0.3072	0.1837
Fed funds, run 2	2	0.25	5	5	0.2968	0.1881
GDP, run 1	5	0.00	4	4	0.2095	0.1423
GDP, run 2	4	0.00	6	4	0.2173	0.1399
GDP, run 3	3	0.00	5	4	0.2131	0.1360
GDP, run 4	3	0.20	11	5	0.2094	0.1305

to the prediction $\hat{Z}(t+1)$ of the next value in the time series, one would obtain an estimate of the error $e(t+1) = Z(t+1) - \hat{Z}(t+1)$. Ideally, the measure should be such that $P(t) = f(e(t+1))$ where f is a known, monotonous function.

Several different predictability measure can be formed. The amount of (local) information in a time series can, for instance, be estimated analytically using random matrix theory, based on the correlation matrix formed from the delay matrix D [8]. In addition, various empirical measures can also be generated, based on the prediction errors obtained in previous time steps. An investigation was made involving both the analytical measure and a few different empirical measures, applied to the rescaled difference series $Z(t)$. However, in all cases, the results were negative, i.e. the proposed predictability measure showed near-zero correlation with the actual prediction error, and therefore these measures will not be described further here.

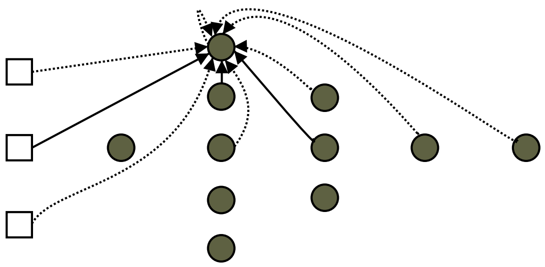

Figure 2: The best evolved network (run 4) for the prediction of the GDP series. Input elements are shown as squares and neurons as filled circles. The neurons are arranged in layers based on their EOT values. For clarity, only the inputs to one neuron are shown. Solid lines indicate positive weights and dotted lines negative ones.

7 Discussion and conclusion

This investigation has shown that it is possible to improve, albeit only slightly, the predictions obtained from standard prediction methods using a generalized version of neural networks (called discrete-time prediction networks, DTPNs) with the possibility of adding a short-term memory through feedback couplings.

In earlier work [2], continuous-time recurrent neural networks were considered for time series prediction. The DTPNs introduced here do not require continuous-time integration, i.e. the network output is obtained by discrete-time equations rather than differential equations, making the evaluation of the networks much faster, while still allowing a rich dynamical structure, including dynamic short-term memory.

The use of an EA for the optimization of the networks removes all restrictions regarding both the behavior of individual neurons as well as the structure of the network as a whole, while still allowing standard feedforward neural networks as a special case.

The importance of structural modifications in the network is illustrated by the fact that, in any given run, the structure of the current best network varied significantly during the run. The final networks often contained rather few neurons and used only a few input elements, illustrating another advantage of using recurrent networks: because of their ability to form a short-term dynamic memory, such networks need not use as many inputs as a feedforward network, thus also reducing the number of networks weights and hence the risk of overfitting.

The best network for prediction of the GDP series, shown in Fig. 2, had a slightly more complex structure. However, in the run generating that network, zero-toggle mutations were used, and indeed the resulting network was far from fully connected, and therefore had, in fact, a somewhat simpler structure than would have been suspected on the basis of the number of neurons involved.

The fact that the predictability measures all gave negative results was expected, and it indicates that the DTPNs really do extract all, or almost all, information available in the time series.

References

[1] Harvey, A., *The econometric analysis of time series*. London School of Economics handbooks in economic analysis, New York; London: Philip Allan, 2nd edition, 1990.
[2] Hulthén, E. & Wahde, M., Improving time series prediction using evolutionary algorithms for the generation of feedback connections in neural networks. *Proc. of Comp. Finance 2004*, 2004.
[3] Giles, C.L., Lawrence, S. & Tsoi, A.C., Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, **44(1/2)**, pp. 161-183, 2001.
[4] Tino, P., Schittenkopf, C. & Dorffner, G., Financial volatility trading using recurrent neural networks. *IEEE-NN*, **12**, pp. 865-874, 2001.
[5] Herbrich, R., Keilbach, M., Graepel, T., Bollmann-Sdorra, P. & Obermayer, K., Neural networks in economics: Background, applications and new developments. *Advances in Computational Economics*, **11**, pp. 169-196, 1999.
[6] Yao, X., Evolving artificial neural networks. *Proc of the IEEE*, **87**, pp. 1423-1447, 1999.
[7] Bäck, T., Fogel, D. & Michalewicz, Z., *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University, 1997.
[8] Ormerud, P., Extracting information from noisy time series data. *Technical report*, Volterra Consulting Ltd, 2004.