Improved time series prediction using evolutionary algorithms for the generation of feedback connections in neural...

2 authors, including:

Erik Hulthén
Chalmers University of Technology
12 PUBLICATIONS **29** CITATIONS

Improved time series prediction using evolutionary algorithms for the generation of feedback connections in neural networks

Erik Hulthén, Mattias Wahde[1]
Department of Machine and Vehicle Systems,
Chalmers University of Technology, Sweden
1. Corresponding author

Abstract

Some results from a method for generating recurrent neural networks (RNN) for prediction of financial and macroeconomic time series are presented. In the presented method, a feedforward neural network (FFNN) is first obtained using backpropagation. While backpropagation is usually able to find a fairly good predictor, all FFNN are limited by their lack of short-term dynamic memory. RNNs, by contrast, may exhibit short-term memory due to feedback connections in the network. In the method presented here, the RNNs are generated by an evolutionary algorithm (EA). The preliminary results indicate that the evolved RNN indeed outperform, by a few per cent, the FFNN obtained through backpropagation on several time series. However, it is noted that, regardless of the predictor used, the prediction error cannot be much improved over that obtained from a very simple predictor. Finally, another approach is tested as well, in which the evolved RNN generate not only a prediction but also a measure of confidence in the prediction.
Key words: time series prediction, evolutionary algorithms, neural networks

1 Introduction

Time series prediction, i.e. the process of predicting future values in a series of data, is of central importance in many fields of science, e.g. meteorology, finance, and macroeconomics. In the physical sciences, it is often possible to derive a sufficiently accurate model of the system, thus reducing the prediction problem to the task of finding the correct parameter values within the framework of the model. However, in finance and macroeconomics a set of fundamental equations that govern the dynamics of the system sufficiently accurately to be useful in time series prediction can rarely be found. Thus, one must instead resort to methods that are model-free or, more accurately, that build their own model. Artificial neural networks (ANN) have been used for economic and macroeconomic time series prediction by several authors, e.g. Dunis *et al* [1], Giles *et al* [2], and Moody [3].

There are several ways of generating ANN for time series prediction. A common method is to use backpropagation to train feedforward neural networks (FFNN) (Haykin [4]). Backpropagation is easy to use, and generally exhibits stable convergence towards small prediction errors. However, FFNNs suffer from an inherent weakness, namely their lack of short-term memory (see Sect. 2). Thus, the selection of the number of input signals (henceforth referred to as the lockback length, L) is crucial in determining the predictions of an FFNN. However, backpropagation gives no information concerning the optimal value of L, and neither does it allow L to vary during training.

In order to overcome the problem caused by the lack of short-term memory in FFNN, and to make the networks more independent of the number of inputs, a recurrent neural network (RNN), i.e. a network with feedback connections, can be used. In principle, an RNN can store information concerning all previous values of a time series, and is therefore not limited by the memory horizon present in FFNNs. On the other hand, the training of RNN is much more complicated. Gradient-based methods, such as backpropagation through time (BPTT) (Werbos [5]) or real-time recurrent learning (RTRL) (Williams *et al* [6]) are applicable in principle, but do not easily accommodate variations in the network architecture during training, which is a crucial drawback since one of the main motivations for using ANNs in the first place is the lack of accurate mathematical models, implying that the training method should allow as much flexibility as possible concerning the network architecture. Furthermore, gradient-based methods require that error signals can be formed, and are thus only applicable to supervised learning. In its simplest formulation, the problem of time series prediction

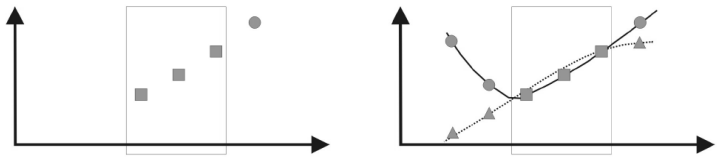

Figure 1: Inherent limitations of FFNN; for a given input (with L elements), an FFNN will always produce the same output.

is a supervised learning problem. However, in more advanced applications, in which the ANN is supposed e.g. to provide not only a prediction but also a measure of its confidence in that prediction, the problem is no longer an example of supervised learning.

Combining ANNs with evolutionary algorithms (EAs), as proposed by Yao [7] and others, is another alternative. EAs are known to be very efficient methods for search and optimization in complex search spaces containing many local optima, in which gradient-based methods, such as backpropagation, are likely to get stuck (Holland [8]). However, evolving ANNs starting from a random network is a very time-consuming process, especially since the crossover operator, which in an EA combines material from two individuals (see below) to form new individuals, is rarely useful. This is an effect of the distributed nature of computation in ANNs: the computation is an emergent property of the entire neural network, and therefore the process of splitting networks and recombining the parts will most often result in a degradation in performance.

Here, an alternative method will be presented, in which FFNN are first trained with backpropagation. Then, the resulting network is mutated slightly to form an initial population for an EA that continues the optimization of the network, transforming it to an RNN by introducing feedback connections if necessary.

2 Method

In this section, the methods used for generating RNNs will be introduced and described.

2.1 Data sets for time series prediction

A time series can be defined as a sequence $X(i), i = 1, \ldots, N$ of time-ordered values, and time series prediction can be defined as the problem of predicting $X(j)$ given $X(j-1), X(j-2), \ldots, X(j-L)$, where L is the lookback length.

A common problem when training ANNs for time series prediction is overfitting, where the training algorithm achieves a very low training error by fine-tuning the weights of the ANN, but at the cost of a reduction in predictive performance, manifested as an increase in the prediction error on new data (i.e. data that were not used during the training procedure). In order to avoid overfitting, and to assess the performance of the ANN in general, it is common to divide the data set into a training set and a validation set (Haykin [4]). The error obtained over the training set is used during training, and the error over the validation set, which is not provided to the training algorithm, is used for determining when to terminate the training.

2.2 Network architectures

If the lookback L is given, the prediction can be performed using two-layer FFNN with L input elements, n_{H} hidden neurons, and one output neuron. The elements of the input signal $\mathbf{x}$ will thus be $x_1 = X(j-1), x_2 = X(j-2)$ etc. and the output signal will be the prediction $\hat{X}(j)$ of $X(j)$. Such an FFNN is shown in the left panel of fig. 2. For the FFNN, the output x^{H} of the hidden neurons is computed as

$$x_i^{\mathrm{H}} = \sigma\left(b_i + \sum_{j=1}^{L} w_{ij}^{\mathrm{I}\to\mathrm{H}} x_j\right), \tag{1}$$

where b_i are the bias terms, $w^{\mathrm{I}\to\mathrm{H}}$ are the weights connecting the input elements to the hidden neurons, and $\sigma(z)$ is the sigmoid function, taken e.g. as $\sigma(z) = 1/(1+\mathrm{e}^{-cz})$, where c is a constant. The output x^O of the network is obtained in a similar way.

However, as mentioned in the introduction, an FFNN has no short-term memory. Consider e.g. a case with $L = 3$, as shown in fig. 1, where two identical inputs $\mathbf{x}^1 = \mathbf{x}^2$ are presented. The FFNN will, in both cases, give exactly the same output, regardless of the signals that preceded the L steps used as inputs. Thus, there is a fundamental limit to the predictive capacity of FFNN, regardless of the training method used. It should be noted that, in macroeconomic and financial time series, identical inputs (for any given L)

do occur, since the variables are often measured with rather low accuracy. For example, unemployment rates are commonly given with a single decimal.

Of course, the problem can be circumvented by increasing L. However, as L is increased, the number of weights in the network grows rapidly, and the risk of overfitting (i.e. fitting the noise in the data) grows with it. Furthermore, the procedure of simply increasing L represents a way to *avoid* a problem rather than *solving* it; the FFNN architecture, lacking short-term memory, is simply not the optimal architecture for prediction problems, and there is no reason to force the data to fit a certain pre-defined architecture. Instead, networks should be provided with short-term memory, and also be allowed to exhibit a flexible architecture, where the size and shape of the network emerges as a result of the training process, rather than being postulated in advance. Both aspects lead to RNNs, an example of which is shown in the right panel of fig. 2.

In network containing feedback connections, the full impact of a change in the input will take some time to reach the output. Thus, using discrete time, updating the output of each neuron only when the input signal changes, will not work. Instead, the RNNs must operate in continuous-time, with new input signals appearing at certain discrete intervals. Thus, for RNNs, the network equations can be taken as

$$\tau_i \dot{x}_i + x_i = \sigma \left(b_i + \sum_{j=1}^{n} w_{ij} x_j + \sum_{j=1}^{L} w_{ij}^{\mathrm{IN}} y_j \right), \; i = 1, \ldots, n, \tag{2}$$

where τ_i are time constants, w_{ij} are interneuron weights connecting the n neurons to each other (note that self-couplings are possible), and w^{IN} are the weights connecting the L input elements to the n neurons. If $\tau_i \to 0$, and if the weights are properly chosen, it is clear that any FFNN can be represented as a special case of an RNN as described by eqn (2).

2.3 Evolutionary algorithms and RNN

There are many different kinds of EAs, e.g. genetic algorithms (GAs), genetic programming (GP) etc. In a GA, a candidate solution to the problem at hand is represented as a fixed-length string of digits known as a chromosome. The chromosome, when decoded, generates an individual (in this case an RNN), which can be evaluated and assigned a fitness score (essentially the inverse of the error) based on its performance. In the evolution of RNNs

for time series prediction, the fitness measure can be taken as $1/e_{\mathrm{RMS}}$, where

$$e_{\mathrm{RMS}} = \sqrt{\frac{1}{N-L+1}\sum_{j=L}^{N}(\hat{X}(j)-X(j))^2} \tag{3}$$

is the RMS error over the training data set.

In a GA, a population consisting of M individuals is maintained. All individuals are evaluated and assigned fitness scores, and new individuals are then formed through the procedures of fitness-proportional selection, crossover, and mutation (i.e. small random variations in the network). The process, which is inspired by darwinian evolution, is repeated until a satisfactory solution to the problem has been found. GAs will not be described in detail in this paper. For detailed information concerning such algorithms, see e.g. Holland [8].

The EA used in this paper differs somewhat from a standard GA. First, it operates directly on the RNNs, rather than going through a decoding-encoding step as described above for the GA. Second, crossover is not used, for the reason mentioned in Sect. 1. Furthermore, three different mutation operators are used: in addition to the ordinary parametric mutations used in any GA (and which, in the case considered here, modify the weights w_{ij} and w_{ij}^{IN}, biases b_i, and time constants τ_i of the RNN), connectivity mutations and structural mutations are used as well. Connectivity mutations add or remove weights between neurons in the network, or between the input elements and the neurons, whereas structural mutations add or remove entire neurons. The parametric mutations are of two kinds: full-range mutations, where the new values is chosen randomly in the full available range (e.g. $[w_{\min}, w_{\max}]$ in the case of network weights), and creep mutations, where the new value is chosen from a narrow probability distribution centered on the old value of the parameter in question.

2.4 Evolving recurrent neural networks

The method for obtaining RNNs for time series prediction operates as follows: First, an FFNN is generated using backpropagation. The number of hidden neurons (n_{H}) and the lockback (L) are chosen using trial-and-error. This is feasible, since the backpropagation algorithm converges quite rapidly. Once a network architecture has been decided upon, a long backpropagation run is performed, to find an FFNN with minimal validation error, as discussed above.

Next, an initial population of RNNs is generated by slightly mutating the FFNN obtained from backpropagation. An exact copy of the FFNN is also included. The FFNN weights are translated to generate appropriate w and w^{IN} matrices. The time between consecutive inputs from the time series is arbitrarily set to 1, and the initial time constants of the RNNs are set to 0.1. With these values, the individual representing an exact copy of the FFNN produces almost identical outputs as the FFNN (the output from the mutated individuals will, of course, differ from that of the FFNN).

Then the EA is allowed to run its course, modifying the values of time constants, biases, connection weights (w_{ij} and w_{ij}^{IN}), as well as adding and removing both connections and neurons. The same training data set is used both when training the FFNN with backpropagation and when evolving RNNs using the EA, and, in both cases, the performance on the validation data set is monitored (but not provided to the training algorithms).

2.5 Benchmark model: Naive strategy

In order to quantify the results from the neural networks a comparison is made with a naive prediction strategy, defined by $\hat{X}(j+1) = X(j)$. This strategy is used as comparison method in e.g. Dunis *et al* [1] and Giles *et al* [2]. Clearly, in order to be useful, a more advanced predictor must, at the very least, outperform this naive strategy.

3 Results

3.1 Improving predictor performance

The procedure described in Sect. 2.4 was applied to several different macroeconomic and financial data sets. In most cases, the evolved RNN outperformed the FFNN generated by backpropagation. Furthermore, the RNN (but *not* the FFNN) generally also outperformed the naive benchmark strategy. However, the improvement in performance was rather small (a few per cent), as shown in Table 1. Due to space limitations, detailed results will only be given for one specific case, namely a prediction of US unemployment.

3.1.1 US unemployment data

The data set used in this case (corresponding to the first row in Table 1) consisted of monthly measurements of the US unemployment rate (seasonally adjusted), from January 1948 to August 2003. The length of the time

Table 1: Results from several different time series. The second and third columns show the length of the training and validation time series, respectively. The fourth column shows the lookback length used in the FFNN, and subsequent columns show the training and validation error (based on ANN output normalized to [0,1]), obtained for the naive strategy, FFNN, and RNN. Series I = US Unemployment data (monthly average), 1948-2003. Series II = Exchange rate USD-JPY (weekly average), 1986-2003.

				Naive		FFNN		RNN	
Series	N_{tr}	N_{val}	L	Tr.	Val.	Tr.	Val.	Tr.	Val.
I	547	118	3	0.0258	0.0163	0.0302	0.0179	0.0286	0.0158
I	442	222	5	0.0275	0.0163	0.0233	0.0161	0.0228	0.0159
II	539	363	4	0.0183	0.0213	0.0191	0.0208	0.0182	0.0206

series was 668. The data were rescaled to the interval [0,1] by first subtracting 2.0 from each data point, and then dividing by 9.0. In this case, L was set to 3, and the best FFNN achieved an RMS training error of 0.0302 and an RMS validation error of 0.0179. The EA ran for 1,300 generations using a population size of 80 individuals. The best RNN (i.e. with lowest validation error) was obtained in generation 1,272. For this network the RMS training error was 0.0286, and the validation error was 0.0158. Scaling back to original units, the RMS validation error from the FFNN was 0.161 percentage points and the RMS validation error from the best RNN was 0.142 percentage points. This can be compared with the naive strategy where the RMS validation error was 0.147 percentage points. The original FFNN and the best RNN are shown in fig. 2. As can be seen from the figure, the EA has added three neurons and a large number of weights. The original network weights were, however, only slightly modified.

3.2 Evolving confidence measures

In the methods considered so far, the ANN predictor attempted to find the best possible predictions over the entire training data set. However, it is common that the performance varies over the data set and, indeed, there is not a priori reason to believe that accurate predictions can be made at every single time step; in a strongly non-linear system, there may be islands of pre-

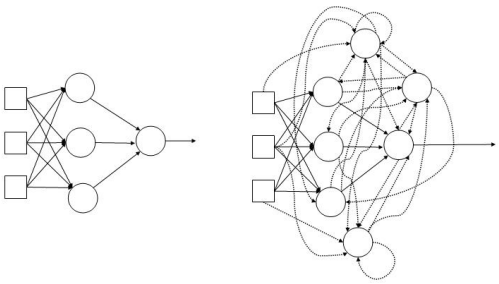

Figure 2: The FFNN (left panel) and the final evolved RNN (right panel) for the prediction of US unemployment (first row in Table 1).

dictability, inserted in a sea of unpredictable or chaotic behavior (Packard [9]). Thus, a predictor (in this case an ANN) that could provide not only a prediction but also a measure of its own *confidence* concerning the prediction, would be very useful. In general, however, the data does not provide a measure of its own predictability (even though estimates can be made based on volatility), and therefore it is impossible to generate an accurate confidence measure using supervised training methods. However, using an EA, it is straightforward: all that is needed is an added neuron whose output is simply taken as the confidence measure. Thus, a modified procedure for evolving RNN was generated, in which all RNNs in the initial generation are supplied with an additional, randomly connected neuron whose output C is defined as the confidence measure. The fitness measure used in the EA is taken as $1/e_{\mathrm{RMS,C}}$ where

$$e_{\mathrm{RMS,C}} = \sqrt{\frac{1}{N-L+1}\frac{\sum_{j=L}^{N}(C(j)(\hat{X}(j)-X(j)))^2}{\sum_{j=L}^{N}C(j)^2}}. \tag{4}$$

The analysis for the US unemployment data was repeated, using the method just described. In general, the EA obtained improvements in the error measure as defined by eqn (4). However, when the ordinary RMS errors were computed, using eqn (3), the results were, in general, slightly worse than for the EA runs without confidence measure. For example, for the US unemployment data, the RMS error over the validation set, for those predic-

tions having confidence $C > 0.90$, was equal to 0.0169. For predictions with $C < 0.10$, the RMS prediction error was equal to 0.0284.

4 Conclusion

In this paper, a method for generating RNNs starting from an FFNN obtained through backpropagation, has been introduced and described. The RNNs obtained with this method outperform the original FFNN as well as a naive benchmark strategy, but only by a small amount. Introduction of an additional neuron providing a confidence measure for the predictions obtained from an RNN had a small negative effect, even though the evolved RNN was able to identify the parts of the time series for which it could make the best predictions. A possible reason for the negative result is the vast increase in complexity when both a prediction and a confidence measure are to be evolved. The issue is currently being further investigated.

References

[1] Dunis, C.L. & Williams, M., Modelling and trading the eur/usd exchange rate: Do neural network models perform better? *Derivatives Use, Trading and Regulation*, **8(3)**, pp. 211–239, 2002.

[2] 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.

[3] Moody, J.E., Economic forecasting: Challenges and neural network solutions. *Proceedings of the International Symposium on Artificial Neural Networks*, Hsinchu, Taiwan, 1995.

[4] Haykin, S., *Neural Networks, A Comprehensive Foundation*. Prentice-Hall, Inc., 2nd edition, 1999.

[5] Werbos, P., Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, **78(10)**, pp. 1550–1560, 1990.

[6] Williams, R.J. & Zipser, D., A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, pp. 270–280, 1989.

[7] Yao, X., Evolving artificial neural networks. *Proc of the IEEE*, **87(9)**, pp. 1423–1447, 1999.

[8] Holland, J., *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, 1975.

[9] Packard, N.H., A genetic learning algorithm for the analysis of complex data. *Complex Systems*, **4(5)**, pp. 543–572, 1990.