Time Series forecasting using machine learning methods

Conference Paper · January 2009

CITATIONS
5

READS
1,562

3 authors:

Michael Stencl
Mendel University in Brno
19 PUBLICATIONS **252** CITATIONS
SEE PROFILE

Ondřej Popelka
Mendel University in Brno
27 PUBLICATIONS **151** CITATIONS
SEE PROFILE

Jiri Stastny
Mendel University in Brno
107 PUBLICATIONS **854** CITATIONS
SEE PROFILE

TIME SERIES FORECASTING USING MACHINE LEARNING METHODS

Michael Stencl, Ondrej Popelka, Jiri Stastny
Department of Informathics
Mendel University of Agriculture and Forestry in Brno
Zemedelska 1/1665, 613 00 Brno
Tel: +420 545 132 728; fax: +420 545 132 245
e-mail: michael.stencl@mendelu.cz, ondrej.popelka@mendelu.cz, stastny@fme.vutbr.cz

ABSTRACT

In this paper we concentrate on prediction of future values based on the past course of that variable, traditionally these are solved using statistical analysis [16] - first a time-series model is constructed and then statistical prediction algorithms are applied to it in order to obtain future values. This paper describes Radial Basis Functions (RBF) Neural Network and Two-level Grammatical Evolution. Both these methods are applied to solve prediction of simplified numerical time series. Sample dataset includes forty generated observations and the goal is to predict five future values.

1 INTRODUCTION

In this paper we concentrate on prediction of future values based on the past course of that variable. Traditionally prediction problems are solved using statistical analysis [13]. As an alternative to statistical methods of time-series modelling we describe two machine learning methods. These can generally be divided into techniques with analytical approach [2] and techniques inspired by biological processes. Most of the biologically inspired techniques are learning methods. Learning methods accept a set of sample data from which they autonomously learn patterns and trends.

2 Radial Basis function Neural Networks

Unlike statistical methods Artificial Neural Networks do not use a separate algorithm for prediction [10], [13]. Forecasting of future values with artificial neural networks is based solely on learned patterns from the input data.

Radial Basis Function neural networks (RBF-NN) belong to the group of feed-forward models of neural networks. A RBF-NN consists of three layers of nodes (Fig. 1). The first is the input layer which transports the input vector to each of the nodes in the hidden (second) layer. Each node of the input layer is connected to each node of the hidden layer. The hidden layer represents data clusters (so-called areas) which form around a node and contain neighbouring nodes in some perimeter. As described in [14] the local units have the relevant output localized to the point in close neighbourhood defined by its parameters. When an input vector arrives on some nodes of the hidden layer simultaneously, each node calculates the distance from the input vector to its own centre [14]. The third layer has only one node which sums the outputs of the hidden layer of nodes to yield the decision value [14].

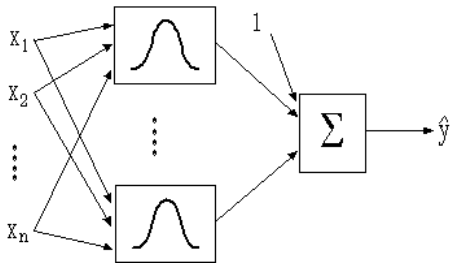

Figure 1: *RBF-NN architecture*

2.1 RBF-NN learning

In the learning process the values of the training set are gradually fed to the neural network, each time a percept is obtained. Since the hidden layer in this network is represented by so-called areas and the middles of the areas are fast discovered, the larger part of the learning process involves only setting of scales and thresholds of the output layer. Gradient method and Last Mean Square (LMS) methods were tested for defining learning error of the neuron network. These gradient methods use relations derived for outgoing layer using Back-propagation (BP) algorithm. The main difference from Back-propagation is that this method optimizes only scales and thresholds of the outgoing layer [11], [12].

Least Mean Square (LMS) method tries to find optimal scaled vector for general middle quadratic error of the network. This scaled vector is given by normal division: $w = \left(H^T H\right)^{-1} H^T y$, where w is scale vector, H is suggestion matrix $H_{ij} = h_j(x_i)$ and y is vector of outgoing values. This method in contrast to others ones uses transient functions of outgoing neurons layer in place of sigmoid linear function [9], [10].

RBF-NN learning is divided into two stages:

1. ***RBF layer neurons learning*** (prototypes learning). In the first stage, prototype *C* and *sigma* are determined for each RBF neuron. For example the algorithms for

cluster analysis can be used. To speed up the first stage, non-adaptive methods can also be used such as uniform or random distribution of RBF neuron centres over the input space.

2. ***Output layer neurons learning***. The objective of the second stage of learning is to determine the weights of input neurons, for example the least square method or gradient algorithms can be used.

Prototypes learning. First the number of clusters in input data is estimated, the pertinence function of model *m* to the cluster is defined and the coordinates of all *p* vectors *Cp* being the centres of clusters are estimated [11].

Both stages includes K-Means algorithm used for powerfull learning of RBF-NN. Steps of K-Means algorithm:

- Initialize RBF neuron centres *C* in random.
- Calculate *m*() for all samples from the training set.
- Calculate new centres *C* as the average of all samples that pertained to centre *k* by the pertinence function.
- Terminate if *m*() does not change, otherwise continue with point 2 [12].

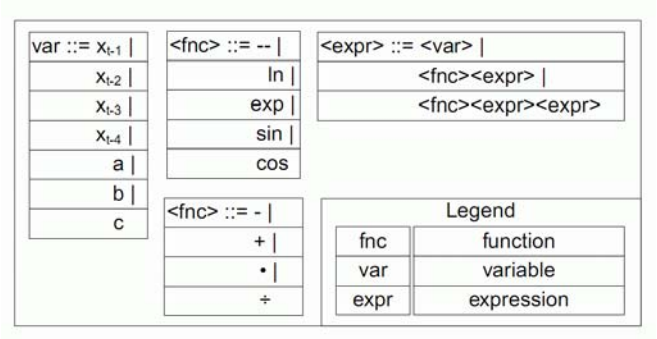

Figure 2: *Production rules for arithmetic expressions*

The centres (prototypes) of neurons are set so that RBF neurons are represented by model clusters for the best. On the other hand, the learning process could be also described as three stages model. In the first stage, centres *cj* are determined for each RBF neuron. The centres *cj* are represented by the weights between the input and the hidden layer. For example the cluster analysis algorithms are used. To speed up this stage, non-adaptive methods can also be used such as the uniform or random distribution of RBF neuron centres over the input space. The second stage setups other values of the RBF neurons. The setup values of the RBF neurons (*bj*) determine the wideness of the area around the estimated centres of *cj*. The objective of the third stage of learning is to determine the weights of the input neurons, for example the least square method or gradient algorithms can be used. In global view, the RBF-NN learning includes unsupervised learning in the first stage. In the second stage the RBF neurons are setup. The typically used function for this setup is the Gaussian Radial Basis Function defined. The RBF neuron determines the important output values in the radial zone with centre in c. The b represents the width of φ and determines the size of the radial zone. The setup parameters of the RBF neuron determine the wideness of the controlled area and affect the generalization capability of the network. If the parameters are smaller means lower generalization capability and on the other hand for wider area the units lost their local mean. In the last stage, the supervised learning is used. The last stage setups the weights wsj. The setup is made by the mineralization process of a typical error function [1].

Description of implementation. Each neuron in the radial basis layer will give the output the value that depends on how close the input vector is to each of the weight vectors of the given neurons. Thus the RBF neurons whose weight vectors are a bit different from input vector *p* have an output close to zero. On the contrary, the RBF neuron whose weight vector is close to the input vector will have a value close to 1. Individual neuron layers have the form of one-dimensional array. The weight matrix is in the form of two-dimensional array, where the index gives the number of neurons being connected. It is necessary to enter the number of RBF neurons *n* for one category.

3 Two-level grammatical evolution

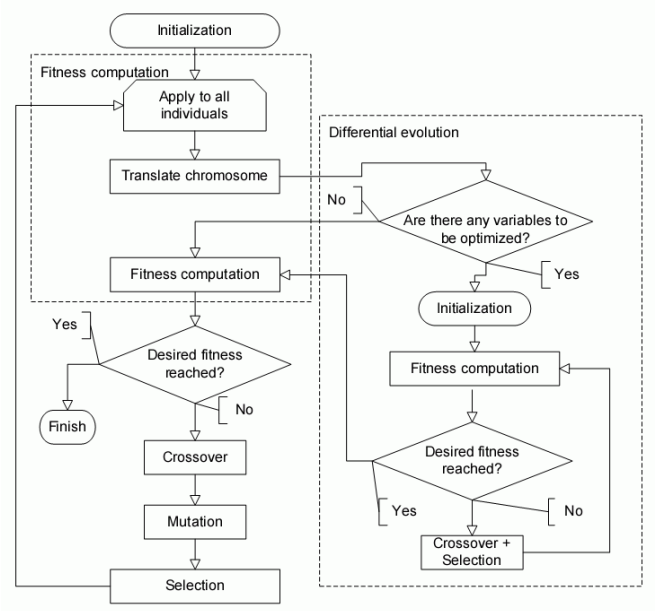

Figure 3: *Two-level grammatical evolution flowchart*

Two-level grammatical evolution comprises of grammatical evolution with backward processing [4] and differential evolution algorithm. Grammatical evolution is based on a genetic algorithm extended with a translation layer inserted between the chromosome and the actual solution. This layer is formed by a processor of context-free grammar which enables the algorithm to create and optimize generic tree structures and retrieve them in arbitrary reusable format defined in a formal language [5].

Table 1: *Definition of grammar used*

$$\Pi = \{expr, fnc, var\}$$

$$\Sigma = \{sin, cos, exp, ln, -, +, *, /, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, a, b, c\}$$

$$S = expr$$

A context-free grammar G is defined as a tuple $G = (\Pi, \sum, P, S)$ where Π is set of non-terminals, Σ is set of terminals, S is initial non-terminal and P is table of production rules. We have chosen to represent the time-series using an autocorrelation model. Definition of the grammar is shown in Table 1, for brevity production rules are shown separately in BNF notation on Figure 2.

Non-terminals are items, which can appear in the individuals' body (that is in the solution) only before or during the translation. Terminals are all symbols which appear in the language generated by the grammar, thus they represent the solution. Start symbol is one non-terminal from the non-terminals set, it is used to initialize the translation process. Production rules define the laws of translation of non-terminals to terminals. The set of non-terminals we used contains arithmetic operators, basic arithmetic functions, variables and constants.

The process of generating a solution is divided into two steps. First an acceptable model is found which represents trends or relations in the data. In the second step the parameters of the model are found and optimized so as to fit the specific dataset. Although grammatical evolution is capable of solving both tasks together [4] it proved to be more flexible to process them separately by using two-level grammatical evolution [6]. For the second step a Differential evolution (DE) algorithm [8] can be used.

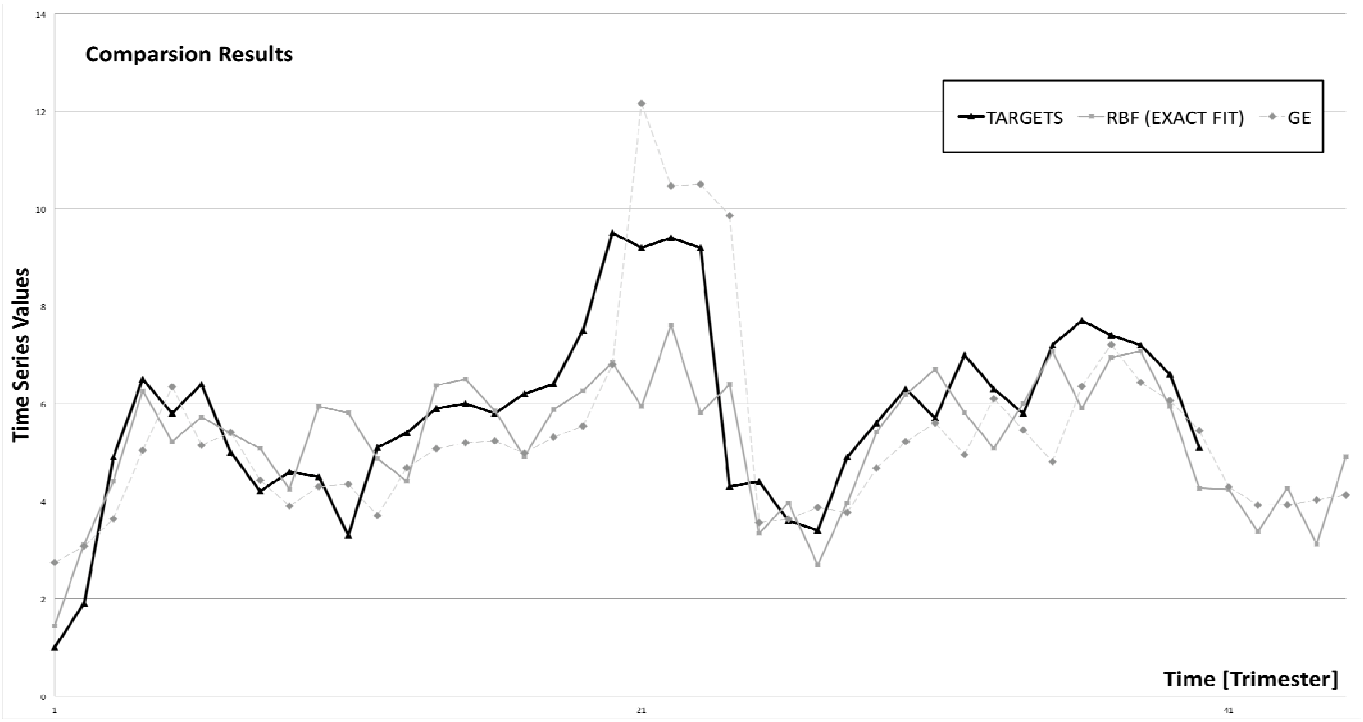

Figure 4: *Comparison of trained RBF-NN (Exact Fit) a two-level grammatical evolution on Target Data*

Similar to grammatical evolution differential evolution is also a modification of genetic algorithm; it is especially suitable for optimizing of real valued parameters of functions since there is no bias in chromosome encoding [7]. This is very favourable for the crossover and mutation operators.

Each individual in DE is represented with a vector $x_{i,G}$, where $i = 0,1,...,NP-1$ and NP is number of individuals in the population. The basic scheme (DE/rand/1) involves generation of a perturbed vector of three randomly chosen vectors. The new vector is generated using the formula $v_{i,G+1} = x_{r1,G} + F(x_{r2,G} - x_{r3,G})$, where $x_{r1,G}$, $x_{r2,G}$ and $x_{r3,G}$ are randomly chosen and different individuals and F is amplification factor of the difference. The new vector v_i is inserted in the population in generation $G+1$ only if it has better fitness value then the original vector.

The first level of the optimization is performed using grammatical evolution. According to the grammar (Figure 2) the output can be a function containing variables (x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}) several symbolic constants (a, b, c). Such function therefore cannot be evaluated and assigned a fitness value. In order to evaluate the generated function a secondary optimization has to be performed to find values for constants.

A flowchart diagram of the two-level optimization is shown on Figure 3. Basically it consists of two nested population loops. The inner loop is using standard differential evolution with DE/rand/1 scheme. The outer loop is a single population of parallel grammatical evolution. This means that in order to evaluate fitness for an individual in GE population it is necessary to create DE population, pass the unknown constants as variable vector to it (each individual can have different number of constants), optimize this vector using DE and substitute optimal values back to GE

individual. The resulting Two-level Grammatical Evolution takes advantage of both the original methods.

6 RESULTS

Sample dataset consists of 40 randomly generated values; values lie in interval <1, 9.5>. First part of the experiment was performed using RBF-NN implementation in Matlab R2007a based on the previously described principles. The dataset was divided into input data and validation data in order to obtain better generalization of the results. Second part of the experiment was realized using our own implementation of two-level grammatical evolution [7]. Grammar used for configuration of the algorithm is described in previous chapter – it is tailored to generate autoregressive formulas of time-series. It is now important to emphasize the difference between both methods. RBF neural network uses fast learning algorithms which adjust the weights in the network so that the network represents trends in the given time-series. Training a RBF neural network is a matter of minutes, but it can provide us only with numeric results. On the other hand Two-level grammatical evolution is a complex algorithm which not only learns the trends in the time-series, it also provides the user with exact description of the time-series. The output of grammatical evolution is both the output data and the formula to obtain them.

Figure 4 shows the input data and two sample runs of both methods. Both methods generally agree on the future values of the time-series. As noted above the output of neural network is only the values displayed, the output of grammatical evolution is both the values displayed and the formula which in this case is

$$\frac{\left((a/36)+x_{t-3}\right)}{x_{t-1}-12}, \qquad (3)$$

where $a = -1324.739$. Therefore Grammatical evolution provides us with more information, but it's training would take much more time than training a neural network. In this simple case it about 10 generations of the underlying genetic algorithm which takes about 40 minutes (about 4 times longer) on the same computer.

Both methods provide comparable results in terms of accuracy. The main difference is in the form in which the results are obtained and in learning performance. Briefly this could be described that it is either possible to obtain numerical results in short time, or use more complicated algorithm to obtain autoregressive formula of the time-series. In future work we would like to use these methods on real-world data with a thorough analysis so that possible inconsistencies in the prediction of both methods can be described and quantified.

Acknowledgements

This work has been supported by the grants:
MSM 6215648904/03 – Research design of MUAF in Brno; MSM 0021630529 – Research design of BUT Brno, Research design of MUAF Brno numbers 116/2102 /IG180651 and 116/2102/IG190611; No 102/07/1503 Advanced Optimisation of Communications Systems Design by Means of Neural Networks GACR.

References

[1] Bishop, C. M., 1991. Improving the generalization properties of radial basis function neural networks. Neural Computation 3, pp. 579–588.

[2] Mitchell, T. M., 1997. Machine Learning. 1st edition. McGraw-Hill. ISBN 0-07-042807-7.

[3] Novak, M., 1998. Umělé neuronové sítě. Teorie a aplikace. C. H. Beck, Praha.

[4] O'Neill, M., Dempsey, I., Brabazon, A., Ryan, C. 2003. Analysis of a Digit Concatenation Approach to Constant Creation In proceedings of the European Conference on Genetic Programming, (EuroGP), Essex, UK, 2003. p. 173-182. ISBN 3-540-00971-X.

[5] O'Neill, M., Ryan, C. 2003. Grammatical Evolution: Evolutionary Automatic Programming. In an Arbitrary Language, Kluwer, 2003, 160 p, ISBN 1-4020-7444-1.

[6] Popelka, O. 2007. Two-level Optimization using Parallel Grammatical Evolution and Differential Evolution, Proceedings of MENDEL'2007, Prague, CR, pp. 88-92. 2007. ISBN 978-80-214-3473-8.

[7] Price, K., 1996. Differential evolution: a fast and simple numerical optimizer. In proceedings of 1996 Biennial Conference of the North American Fuzzy Information Processing Society, NAFIPS, pp. 524-527, IEEE Press, New York, 1996. ISBN:0-7803-3225-3.

[8] Price, K. V., Storn, R. M., Lampinen, J. A. 2005. Differential evolution-a practical approach to Global Optimization. Springer, 543 p., ISBN 3-540-20950-6.

[9] Ripley, B. D., 1996. Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge.

[10] Sarle, W. S., 1994. Neural Networks and Statistical Models. Proceedings of the Nineteenth Annual SAS Users Group International Conference, Cary, NC: SAS Institute, pp 1538–1550.

[11] Stastny, J., Skorpil, V., 2005. Neural Networks Learning Methods Comparison. International Journal WSEAS Transactions on Circuits and Systems, Issue 4, Volume 4, pp. 325–330, ISSN 1109-2734.

[12] Stastny, J., Skorpil, V., 2007. Genetic Algorithm and Neural Network. WSEAS Applied Informatics & Communications, pp. 347–351, ISBN 978-960-8457-96-6, ISSN 1790-5117.

[13] Tseng, F. M., Yu, H. C., Tzeng, G. H., 2002. Combining neural network model with seasonal time series ARIMA model. Technological Forecasting and Social Change, 69, 71–87.

[14] Wedding D. K., Cios K. J., 1996. Time series forecasting by combining RBF networks, certainty factors, and the Box-Jenkins model. Neurocomputing, vol. 10. pp. 149–168. DOI:10.1016/0925-2312(95)00021-6.