Generalized Regression Neural Network for Forecasting Time Series with Multiple Seasonal Cycles

Grzegorz Dudek

Department of Electrical Engineering, Czestochowa University of Technology,
Al. Armii Krajowej 17, 42-200 Czestochowa, Poland

dudek@el.pcz.czest.pl

Abstract. This paper presents a method of forecasting time series with multiple seasonal cycles based on Generalized Regression Neural Network. This is a memory-based, fast learned and easy tuned type of neural network. The time series is preprocessed to define input and output patterns of seasonal cycles, which simplifies the forecasting problem. The method is useful for forecasting nonstationary time series with multiple seasonal cycles and trend. The model learns with the help of differential evolution or simple enumerative method. The performance of the proposed method is compared with that of other forecasting methods based on Nadaraya-Watson estimator, neural networks, ARIMA and exponential smoothing. Application examples confirm valuable properties of the proposed method and its highest accuracy among the methods considered.

Keywords: Seasonal time series forecasting, generalized regression neural network, differential evolution, pattern similarity based forecasting, short-term load forecasting.

1 Introduction

Many time series exhibit multiple seasonal cycles of different lengths. Good example of such a time series is hourly electricity demand in Poland presented in Fig. 1. This time series has three seasonal periods: daily, weekly and annual. The daily and weekly profiles change during the year. The daily profile depends on the day of the week as well. This time series expresses trend and is nonstationary in mean and variance. These all features have to be captured by the flexible forecasting model.

A variety of methods have been proposed for forecasting seasonal data. The most commonly employed classical models are exponential smoothing (ES) and seasonal autoregressive moving average models (ARMA). In ES the time series is modeled using a set of equations expressing level, growth and each seasonal cycle. These components can be combined additively or multiplicatively. Examples of using ES for forecasting multiple seasonal data can be found in [1] and [2]. One of the drawback of ES is overparameterization which involves initialization and updating of a large number of terms.

adfa, p. 1, 2011.

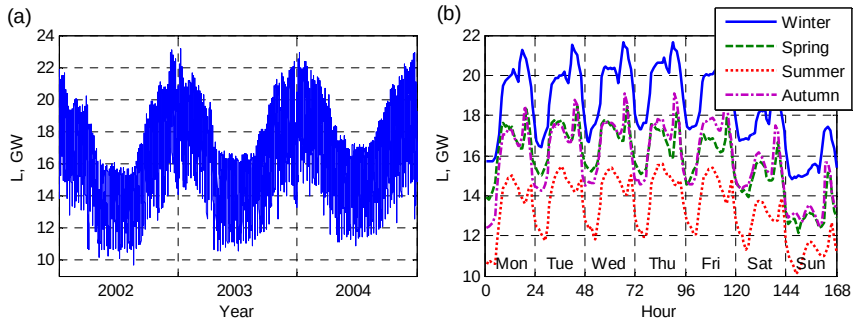

Fig. 1. The hourly electricity demand in Poland in three-year (a) and one-week (b) intervals.

A base ARMA model can be extended for fitting a time series with a trend and seasonal variations. This consists in appropriate differentiation of a time series. The seasonal ARIMA can only deal with time series that are stationary in variance. When the variance changes in time transformation of a time series is needed. There are a very large number of ARIMA models. The selection process of ARIMA model and its order is usually considered subjective and difficult to apply. Another disadvantage of ARIMA is the linear character of the model.

The rapid development of computational intelligence and machine learning brings new tools for forecasting. They include mainly artificial neural networks (ANNs), fuzzy logic and intelligent searching methods, such as evolutionary algorithms and swarm intelligence. ANNs are very attractive as nonlinear methods of forecasting due to their universal approximation property, massive parallelism among a large number of simple units, learning capabilities, robustness in the presence of noise, and fault tolerance. Using fuzzy logic we can enter uncertain and imprecise data to the model. The forecasting model is composed of the if-then rules which can be formulated verbally by experts or extracted from data in the learning process. Evolutionary algorithms and related methods are useful for model optimization and learning. In a stochastic searching process they are able to get out of the local optima and find better solutions.

This paper focuses on the design of a forecasting method for time series with multiple seasonal cycles based on General Regression Neural Network. This is memory-based locally weighted regression method. The integral part of this model is data preprocessing and defining patterns of seasonal cycles of time series as explanatory and response variable vectors. Patterns simplify the forecasting problem filtering out the trend and redundant seasonal variations of periods longer than the basic one. The nonstationarity in mean and variance is also eliminated. The parameters of the GRNN model are estimated using differential evolution or simpler enumerative method.

2 Patterns of the Seasonal Cycles

The input patterns representing explanatory variables are defined as the transformed time series elements taken from basic seasonal cycles (seasonal cycle of the

shortest length). The input pattern $\mathbf{x}_i = [x_{i,1}\ x_{i,2}\ \ldots\ x_{i,n}]$ maps the time series elements from the i-th cycle $\mathbf{z}_i = [z_{i,1}\ z_{i,2}\ \ldots\ z_{i,n}]$ as follows:

$$x_{i,t} = \frac{z_{i,t} - \bar{z}_i}{\sqrt{\sum_{l=1}^{n} (z_{i,l} - \bar{z}_i)^2}}, \tag{1}$$

where: $i = 1, 2, \ldots, M$ – the seasonal cycle number, $t = 1, 2, \ldots, n$ – the time series element number in the cycle i, $z_{i,t}$ – the t-th time series element in the cycle i, $\bar{z}_i$ – the mean value of elements in the seasonal cycle i.

Definition (1) expresses normalization of the vectors $\mathbf{z}_i$. After normalization they have unity length, zero mean and the same variance. Thus the nonstationary time series $\{z_j\}$ is represented by x-patterns having the same mean and variance.

Similarly the output patterns $\mathbf{y}_i = [y_{i,1}\ y_{i,2}\ \ldots\ y_{i,n}]$ maps the elements from the seasonal cycle $i+\tau$: $\mathbf{z}_{i+\tau} = [z_{i+\tau,1}\ z_{i+\tau,2}\ \ldots\ z_{i+\tau,n}]$, where $\tau > 0$ is a forecast horizon:

$$y_{i,t} = \frac{z_{i+\tau,t} - \bar{z}_i}{\sqrt{\sum_{l=1}^{n} (z_{i,l} - \bar{z}_i)^2}}. \tag{2}$$

Note that vectors $\mathbf{y}$ are defined using known current process parameters ($\bar{z}_i$ and the square root in the denominator as a measure of dispersion of elements in i-th seasonal cycle). This enables us to determine the forecast of vector $\mathbf{z}_{i+\tau}$ using (2) after the forecast of pattern $\mathbf{y}$ is calculated by the model.

The input and output patterns are paired and included in the set $L = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_M, \mathbf{y}_M)\}$ from which the training sets are generated. The pairs $(\mathbf{x}_i, \mathbf{y}_i)$ represent the seasonal periods between which the distance in time is τ. The model learns the mapping $\mathbf{x} \rightarrow \mathbf{y}$ and then it forecasts the output pattern corresponding to the input pattern $\mathbf{x}$ (query pattern) which is presented to the model.

3 Generalized Regression Neural Network

Generalized Regression Neural Network (GRNN) is a kind of Radial Basis Function (RBF) neural network with a one pass learning algorithm and highly parallel structure. GRNN was introduced by Specht in 1991 [3] as a memory-based network that provides estimates of continuous variables. The algorithm provides smooth approximation of a target function even with sparse data in a multidimensional space. The advantages of GRNN are fast learning and easy tuning. The GRNN is composed of four layers: input, pattern (radial basis layer), summation and output as shown in Fig. 1.

Each neuron of the pattern layer uses a radial basis function as an activation function. This function is commonly taken to be Gaussian:

$$G_j(\mathbf{x}) = \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{C}_j\right\|^2}{s_j^2}\right), \tag{3}$$

where: $\mathbf{C}_j$ is a center vector, s_j is a smoothing parameter or bandwidth and ||.|| is the Euclidean norm.

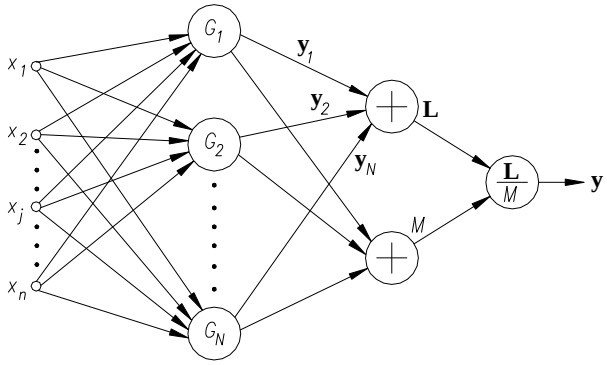

Fig. 2. GRNN architecture.

Each training vector is represented by one pattern neuron with the center $\mathbf{C}_j = \mathbf{x}_j$, $j = 1, 2, \ldots, N$, where N is a number of training points. The neuron output expresses the similarity between the input vector $\mathbf{x}$ and the j-th training vector. So the pattern layer maps the n-dimensional input space into N-dimensional space of similarity. The GRNN output is an average of training y-patterns weighted by the degree of similarity between paired with them x-patterns and the query pattern:

$$\hat{\mathbf{y}} = g(\mathbf{x}) = \frac{\sum_{j=1}^{N} G_j(\mathbf{x})\mathbf{y}_j}{\sum_{j=1}^{N} G_j(\mathbf{x})}. \tag{4}$$

Note that the GRNN generates a vector as an output. The dimension of this vector does not affect the number of parameters to estimate unlike in other popular models such as multilayer perceptron or neuro-fuzzy networks. This should be considered as a valuable property.

The performance of GRNN is related with bandwidths s_j governing the smoothness of the regression function (4). Determining optimal bandwidth values is a major problem in GRNN training. In Section 4 we propose the differential evolution algorithm for estimation of bandwidths.

The forecasting model similar to GRNN called Nadaraya-Watson estimator was presented in [4]. In this estimator the product kernel is used as RBF. The product kernel has different bandwidths for each component of $\mathbf{x}$. But for the different training patterns the same set of bandwidths are used. In the case of GRNN with Gaussian

functions for each training pattern there is only one bandwidth but for each pattern the bandwidth is different.

4 Differential Evolution for GRNN Training

The bandwidth values s_j are tuned using the differential evolution (DE). This is an heuristic algorithm for global optimization over continuous spaces [5]. It is considered as the one of the most powerful stochastic optimization algorithm [6]. Unlike traditional evolutionary algorithms, DE employs difference of the solution vectors to explore the solution space.

In our case the solution vector is of the form $\mathbf{s} = [s_1, s_2, \ldots, s_N]$. The initial population of vectors is usually chosen randomly. New populations are generated in DE using mutation and crossover operators. During mutation for each population vector a mutant $\mathbf{v}_i$ is formed by adding the weighted difference between two population vectors to a third vector:

$$\mathbf{v}_i = \mathbf{s}_{r_1} + F(\mathbf{s}_{r_2} - \mathbf{s}_{r_3}), \tag{5}$$

where: $r_1, r_2, r_3 \in \{1, 2, \ldots, S\} \backslash i$ are random indexes (different from each other), S is a population size and $F \in [0, 2]$ is a coefficient controlling the mutation range.

The i-th vector is combined with its mutant using discrete crossover:

$$s'_{i,j} = \begin{cases} v_{i,j}, & \text{if } \xi_{i,j} \le CR \text{ or } j = \zeta_i, \\ s_{i,j}, & \text{if } \xi_{i,j} > CR \text{ and } j \ne \zeta_i, \end{cases} \tag{6}$$

where: $j = 1, 2, \ldots, N$, $\xi_{i,j}$ is a random number from uniform distribution $U(0, 1)$, ζ_i is a randomly chosen index from $\{1, 2, ..., N\}$ which ensures that the new solution gets at least one component of the mutant $\mathbf{v}_i$, and $CR \in [0, 1]$ is a crossover constant.

The trial solution $\mathbf{s}'_i = [s'_{i,1}, s'_{i,2}, \ldots, s'_{i,N}]$, created as a result of mutation and crossover, replaces its parent solution $\mathbf{s}_i$, if the cost function value is smaller for $\mathbf{s}'_i$ than for $\mathbf{s}_i$. After repeating this mutation, crossover and selection procedures for $i = 1, 2, ..., S$ we get a new population which is processed in the same way.

This is the basic strategy of DE. There are three control parameters here: the population size S, the mutation scale factor F and the crossover constant CR. The effect of each of these parameters on the performance of DE is discussed in [6]. The inventors of the algorithm recommend the value of S between $5n$ and $10n$, and $F = 0.5$.

The parameter CR controls how many components in expectation are changed in a solution vector. For small CR few components are changed and the stepwise movement tends to be orthogonal to the current coordinate axes. Higher CR value causes most of the mutant components to be inherited preventing this effect. In the application examples described in the next section the values of F and CR were tuned experimentally.

5 Application Examples

In this section we use the hourly electricity demand time series to test our GRNN model and compare it with other popular models used for forecasting time series with multiple seasonal cycles. The analyzed time series is shown in Fig. 1. (This data can be downloaded from the website http://gdudek.el.pcz.pl/varia/stlf-data.)

Our goal is to forecast one seasonal (daily) period ahead ($\tau = 1$) for January (without untypical 1 January) and July 2004. Thus there is 30 + 31 = 61 forecasting tasks. For each forecasting task (test sample) the learning set is prepared individually from the historical data. It contains pairs of patterns from the set L representing the same days of the week (Monday, ..., Sunday) as the query pattern and forecasted **y** pattern. For each forecasting task the separate GRNN model is created and learned using DE. The solutions generated in DE are evaluated in the local leave-one-out procedure, in which the validation samples are chosen one by one from the set of 12 nearest neighbors of the query pattern.

In the first part of the study we investigate the efficiency of DE at different values of its parameters:

- DE1: $CR = 0.1$, $F = 0.5$,
- DE2: $CR = 0.3$, $F = 0.5$,
- DE3: $CR = 0.9$, $F = 0.5$,
- DE4: $CR = 0.1$, $F = 0.1$,
- DE5: $CR = 0.1$, $F = 1.0$,
- DE6: $CR = 0.1$, $F = 2.0$.

The population size in all variants was constant $S = 210$.

The convergence curves for DE with different parameter values in Fig. 3 are presented. The DE2 and DE3 variants with higher CR value than DE1 converge slower than DE1. The best value of F causing the fastest convergence turned out to be 0.1. However the improvement on the validation samples are observed the test error was not reduced. Therefore, the simpler method for selection of the bandwidths was used as follows. The distance d_5 between the query point and its 5-th nearest neighbor in the training set is determined. It is assumed that the bandwidth for all neurons is equal to $a \cdot d_5$, where a is a discrete parameter tuned by enumerating. The value of a ensuring the best results on validation samples was 0.5.

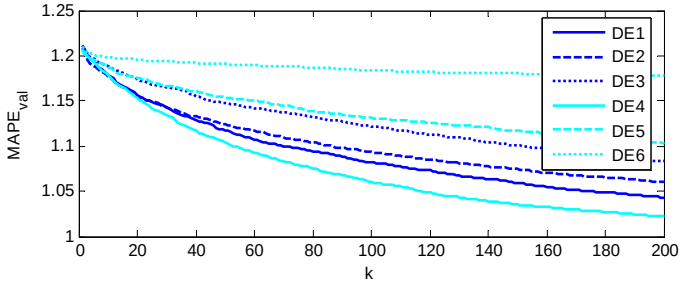

Fig. 3. Comparison of convergence curves for different DE variants.

We compare the proposed GRNN model with the model based on Nadaraya-Watson estimator (N-WE), two neural network models: RBF and multilayer perceptron (MLP) and two classical models: ARIMA and ES. N-WE model is described in [4], ARIMA, ES and MLP models are described in [7]. Just as GRNN the models based on N-WE, RBF and MLP use patterns of seasonal cycles as explanatory and response variable vectors. In N-WE estimator the bandwidths were determined using the Scott's rule [4], depending on the sample standard deviation, sample size and dimension. RBF neural network was designed in Matlab using newrb function. The bandwidth of radial basis functions (the same for all neurons) was set to the distance between the query point and its k-th nearest neighbor in the training set. On the basis of preliminary experiments it was assumed $k = 12$. The MLP was learned in the local learning procedure in which the training samples were selected from the neighborhood of the query pattern. The one-neuron model was selected as an optimal solution. To find the best ARIMA and ES models automated procedures implemented in the forecast package for the R system [8] were used.

The results of forecasting: mean absolute percentage errors for the test samples $MAPE_{tst}$ and their interquartile ranges (IQR) in Table 1 are shown. For comparison the results determined using the naïve method are also shown. The forecast rule in this case is as follows: the forecasted daily cycle is the same as seven days ago.

Table 1. Results of forecasting.

Model	January		July		Mean	
	$MAPE_{tst}$	IQR	$MAPE_{tst}$	IQR	$MAPE_{tst}$	IQR
GRNN	1.21	1.22	0.90	0.93	1.05	1.04
N-WE	1.23	1.19	0.88	0.87	1.05	1.09
RBF	1.56	1.41	1.10	1.18	1.33	1.30
MLP	1.32	1.30	0.97	1.01	1.14	1.15
ARIMA	2.64	2.34	1.21	1.24	1.91	1.67
ES	2.35	1.88	1.19	1.30	1.76	1.56
Naïve	6.37	5.36	1.29	1.20	3.78	3.82

The best forecasts were obtained by GRNN and N-WE. These two methods gave statistically indistinguishable errors (Wilcoxon signed-rank test was used). A little worse results were obtained using MLP. The worst results were achieved using classical models: ARIMA and ES. This is probably due to the learning sample used for estimation of parameters in ARIMA and ES, which contains the time series fragment (12-week fragment in our case) directly preceding the forecasted fragment. The irregularities in this preceding fragment affect adversely the forecasts. In the proposed approach the learning set is not limited to the fragment preceding the forecast but is composed of the most similar patterns to the query pattern. So irregular patterns, which are not similar to the query pattern are not included in this set, unless their irregularities correspond to the irregularities in the query pattern.

Note that GRNN model optimized using enumeration method has only one parameter to estimate (a). This is a great advantage. Such a model is easy to optimize and has good generalization properties.

6 Conclusions

The memory-based models operating on patterns of seasonal cycles: GRNN and N-WE turned out to be the most accurate in forecasting time series with multiple seasonal cycles compared to RBF and MPL neural networks and classical statistical models: ARIMA and ES. The memory-based approaches do not estimate a global model but defer the processing of data until a forecast is requested. The forecast is derived from the neighborhood of the query point using locally weighted regression. A key problem here is the selection of appropriate weighting functions to get the best generalization performance given a set of sparse and noisy data. In the application example an exact estimation of bandwidth values for each neuron using differential evolution did not bring an expected reduction in the test error. This is probably due to the sparse data: there is not enough training data in the neighborhood of the query pattern to build an accurate local model. The simpler enumeration method for bandwidth estimation where we are searching for the value the a coefficient provided better results. In this case GRNN has only one parameter to estimate. The instant training and easy tuning are great advantages of GRNN.

The proposed forecasting model owes its good performance not only to the valuable properties of GRNN but also to the initial transformations of data and appropriate definitions of patterns of seasonal cycles. The patterns simplify a forecasting problem eliminating nonstationarity of time series in mean and variance and removing the trend and seasonal variations of periods longer than the basic one.

Acknowledgment. The study was supported by the Research Project N N516 415338 financed by the Polish Ministry of Science and Higher Education.

Literature

1. Taylor J.W.: Exponentially Weighted Methods for Forecasting Intraday Time Series with Multiple Seasonal Cycles. International Journal of Forecasting 26, 627–646 (2010).
2. Gould P.G. et al.: Forecasting Time-Series with Multiple Seasonal Patterns. European Journal of Operational Research 191, 207–222 (2008).
3. Specht D.F.: A General Regression Neural Network. IEEE Transactions on Neural Networks 2(6), 568–576 (1991).
4. Dudek G.: Short-term Load Forecasting Based on Kernel Conditional Density Estimation. Przegląd Elektrotechniczny (Electrical Review) 86 (8), 164–167 (2010).
5. Storn R., Price K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11 (4), 341–359 (1997).
6. Das S., Suganthan P.N.: Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation 15(1), pp.4–31 (2011).
7. Dudek G.: Forecasting Time Series with Multiple Seasonal Cycles using Neural Networks with Local Learning. In: Rutkowski L. et al. (eds.): Artificial Intelligence and Soft Computing, ICAISC 2013, LNCS 7894, 52–63 (2013).
8. Hyndman, R.J., Khandakar, Y.: Automatic Time Series Forecasting: The Forecast Package for R. Journal of Statistical Software 27(3), 1–22 (2008).