Neurocomputing 205 (2016) 64–74

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Neural networks for pattern-based short-term load forecasting: A comparative study

Grzegorz Dudek

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

ARTICLE INFO

Article history:
Received 30 September 2015
Received in revised form
29 February 2016
Accepted 16 April 2016
Communicated by Ngoc Thanh Nguyen
Available online 17 May 2016

Keywords:
Neural networks
Patterns of seasonal cycles
Short-term load forecasting
Time series

ABSTRACT

In this work several univariate approaches for short-term load forecasting based on neural networks are proposed and compared. They include: multilayer perceptron, radial basis function neural network, generalized regression neural network, fuzzy counterpropagation neural networks, and self-organizing maps. A common feature of these methods is learning from patterns of the seasonal cycles of load time series. Patterns used as input and output variables simplify the forecasting problem by filtering out a trend and seasonal variations of periods longer than a daily one. Nonstationarity in mean and variance is also eliminated. In the simulation studies using real power system data the neural network forecasting methods were tested and compared with other popular forecasting methods such as ARIMA and exponential smoothing. The best results were achieved for generalized regression neural network and one-neuron perceptron learned locally.

1. Introduction

Short-term load forecasting (STLF) is essential for power system control and scheduling. Load forecasts of short horizon (from minutes to days) are necessary for energy companies to make decisions related to planning of electricity production and transmission, such as unit commitment, generation dispatch, hydro scheduling, hydro-thermal coordination, spinning reserve allocation, interchange and low flow evaluation. Electricity markets also require the precise load forecasts because the load is the basic driver of electricity prices. The forecast accuracy translates to financial performance of energy companies (suppliers, system operators) and other market participants and financial institutions operating in energy markets.

Neural networks are widely used in STLF due to their flexibility which can reflect process variability in uncertain dynamic environment and complex relationships between variables. The main drivers of the power system load are: weather conditions (temperature, wind speed, cloud cover, humidity, precipitation), time, demography, economy, electricity prices, and other factors such as geographical conditions, consumer types and their habits. The relationships between explanatory variables and power system load are often unclear and unstable in time. In this work we focus on univariate forecasting methods, in which only historical load time series is used as input to predict the future values of power system load.

The load time series contains a trend, multiple seasonal variations and a stochastic irregular component. From Fig. 1, where the hourly electrical load of the Polish power system is shown (these data can be downloaded from the website http://gdudek.el.pcz.pl/varia/stlf-data), it can be observed annual, weekly and daily cycles. A daily profile, on which we focus in STLF, depends on the day of the week and season of the year. Moreover, it may change over the years. The noise level in a load time series depends on the system size and the customer structure. A trend, amplitude of the annual cycle, weekly and daily profiles and noise intensity may change considerably from dataset to dataset.

Due to the importance of STLF and the problem complexity many various STLF methods has been developed. They can be roughly classified as conventional and unconventional ones. Conventional STLF approaches use regression methods, smoothing techniques and statistical analysis. The most commonly employed conventional models are: the Holt-Winters exponential smoothing (ES) and the autoregressive integrated moving average (ARIMA) models [1]. In ES the time series is decomposed into a trend component and seasonal components which can be combined additively or multiplicatively. ES can be used for nonlinear modeling of heteroscedastic time series, but the exogenous variables cannot be introduced into the model. The most important disadvantages of ES are overparameterization and a large number of

E-mail address: dudek@el.pcz.czest.pl
URL: http://www.gdudek.el.pcz.pl

http://dx.doi.org/10.1016/j.neucom.2016.04.021
0925-2312/

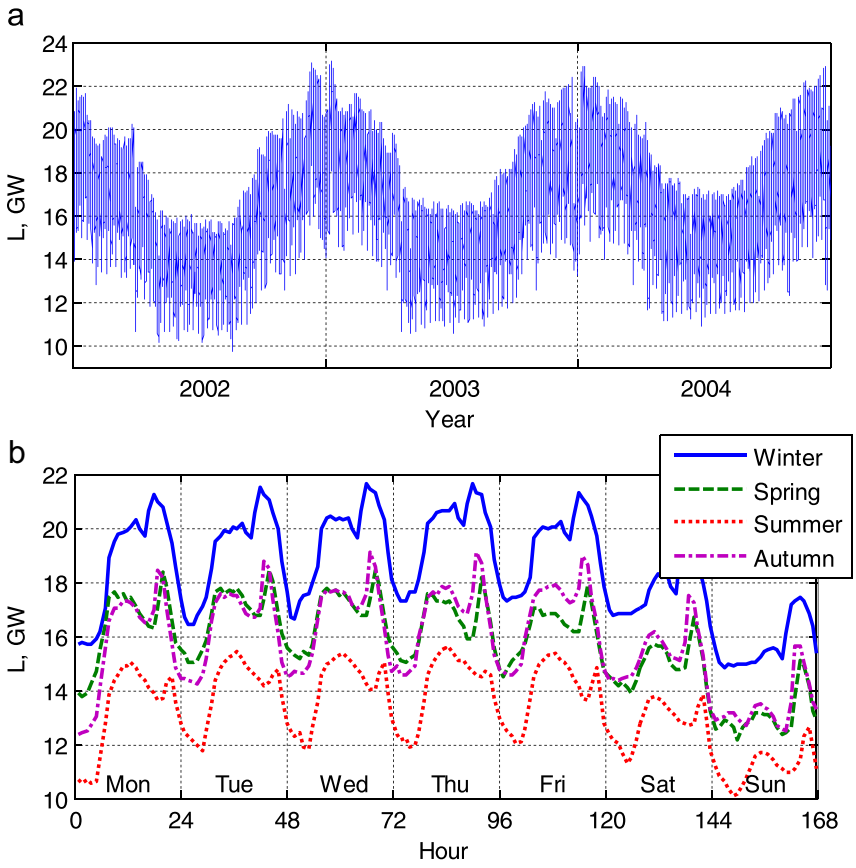


Fig. 1. The hourly electricity demand in Poland in three-year (a) and one-week (b) intervals.

starting values to estimate. Recently developed exponentially weighted methods in application to STLF are presented in [2].

ARIMA processes are well suited to express the stochastic nature of the load time series. Modeling of multiple seasonal cycles as well as introducing exogenous variables is not a problem in ARIMA. The disadvantage of ARIMA models is that they are able to represent only linear relationships between variables. The difficulty in using ARIMA is the problem of order selection which is considered to be subjective. To simplify the forecasting problem the time series is often decomposed into a trend, seasonal components and an irregular component. These components, showing less complexity than the original series, are modeled independently (see [3]). Another decomposition way based on wavelet transform is described in [4].

Unconventional STLF approaches employ new computational methods such as artificial intelligence and machine learning ones. These approaches are reviewed in [5,6]. They include: neural networks (NNs), fuzzy inference systems, neuro-fuzzy systems, support vector machines and ensembles of models. Among them the most popular are NNs. They have many attractive features, such as: universal approximation property, learning capabilities, massive parallelism, robustness in the presence of noise, and fault tolerance. But there are also some drawbacks of using NNs: disruptive and unstable training, difficulty in matching the network structure to the problem complexity, weak extrapolation ability and many parameters to estimate (hundreds of weights). These issues with NNs as well as problems with load time series complexity are addressed in STLF literature in various ways. For example in [7] the Bayesian approach is used to control the multilayer perceptron (MLP) complexity and to select input variables. As inputs are used: lagged load values (unprocessed), weather variables, and dummies to represent the days of the week and calendar variables. In [8] the load time series is decomposed using wavelet transform into lower resolution components (approximation and details). Each component is predicted by MLP using gradient-based algorithm. After learning the MLP weights are adjusted using evolutionary algorithm to get result nearer to the optimal one. Similar decomposition of the load time series using the wavelet transform for extraction relevant information from the load curve was used in [9]. A particle swarm optimization algorithm was employed to adjust the MLP weights. In [10] a generic framework combining similar day selection, wavelet decomposition, and MLP is presented. The MLP is trained on the similar days which are first decomposed using Daubechies wavelets. The similar day is a day which has the same weekday index, day-of-a-year index and similar weather to that of tomorrow (forecasted).

Another popular NN architecture used for STLF is radial basis function NN (RBFNN). It approximates the relationship between explanatory variables and load by a linear combination of radial basis functions (usually Gaussian), which nonlinearly transform the input data. Comparing to MLP the learning algorithm for RBFNN is simpler (there is no need for laborious error back-propagation). In [11] a hybrid system for STLF is described, where RBFNN forecasts the daily load curve based on historical loads. The RBFNN weights are optimized using genetic algorithm. Then the Mamdani-type fuzzy inference system corrects the forecast depending on the errors and maximal daily temperature. RBFNN is employed in [12] to forecast loads based on historical loads and historical and forecasted temperatures. Then the neuro-fuzzy system corrects the forecast depending on changes in electricity prices. In the approach described in [13] the load time series is decomposed into five components using wavelet transform.

Low-frequency component is predicted using RBFNN, while the high-frequency components (details) are predicted by averaging the details of few days of the same type as the day of forecast.

A self-organizing map (SOM) is another popular NN used in STLF. This network is trained using unsupervised competitive learning to produce a low-dimensional representation of the input space. Input vectors are grouped and represented by neurons. The hierarchical forecasting method composed of two SOMs is presented in [14]. One SOM maps the input vector composed of the past loads and indicators of forecasted hour into the vector of distances between neurons and the winning neuron. This vector is an input for the second SOM, which maps it to the forecasted load at hour t. The forecasts of loads for the next hours are obtained recursively. In [15] SOM is used for grouping the variable vector carrying information about the short-term dynamics of the load time series. These variables include load characteristic points, such as peaks, selected weather factors, and indicators of a day of the week and a holiday. Then, for each group the regression model was built using support vector regression with selected historical loads and temperatures as inputs. SOM is applied in [16] to split the historical data dynamics into clusters. A flexible smooth transition autoregressive model is used to determine the forecast based on the linear combination of autoregressive models identified for each cluster. The weights in this combination are determined dynamically using MLP. SOM with a planar layer of neurons is used in [17] for grouping the standardized daily load curves. Observing the responses of neurons for the successive training samples, the probabilities of transitions between neurons are determined. The forecast of the standardized load curve is calculated from the weights of these neurons for which the probability of transition from the winner neuron is non-zero. To get the forecast of the real loads, the forecasts of the daily load curve mean and standard deviation are needed. For this purpose, the authors use RBFNN. In [18] the vector of predictors is expanded by the additional component: the forecasted variable. Such vectors are grouped using SOM. After learning, vector of predictors is presented (without additional component) and recognized by the winning neuron. The weight of the winner corresponding to the additional component of the input training vector is the forecast. The predictors are historical loads projected into lower dimensional space using curvilinear component analysis.

Many other types of NNs have been used for STLF including: recurrent NNs, generalized regression NNs, probabilistic NNs, adaptive resonance theory NNs, functional link NNs and counterpropagation NNs. The survey of NN applications to STLF can be found in [19,20].

In this work a specific way of data preprocessing is used to simplify the STLF problem. We create patterns which contain information about the shapes of daily curves. Input and output patterns are defined which represent two types of daily periods. The input pattern represents the daily period preceding the forecasted one and the output pattern represents the forecasted daily period. Due to representing multiple seasonal time series using patterns we eliminate nonstationarity in input data, and we remove the trend and seasonal cycles of periods longer than the daily one. To model the relationship between input and output patterns we use NNs of different architectures. Each NN is optimized on a set of patterns which are similar to the query pattern (i.e. the input pattern for which we want to get the forecast). Thus the NN is optimized locally in the neighborhood of the query pattern (NN is locally competent). This approach maximizes the model quality for particular query pattern. For another query pattern the NN is learned and optimized again.

The rest of this paper is organized as follows. In Section 2 we define patterns of daily cycles of the load time series. Section 3 presents neural models for STLF. The learning and optimization procedures for each model are given. In Section 4 we test the neural models on real load data. We compare results of the proposed methods to other STLF methods. Finally, Section 5 concludes the paper. At the end of the article the list of symbols is presented.

2. Data preprocessing and patterns

A time series of electrical load expresses a trend, which is usually non-linear, and three seasonal cycles: annual, weekly and daily ones (see Fig. 1). It is nonstationary in mean and variance. Data preprocessing by defining patterns simplifies the forecasting problem of the time series with multiple seasonal cycles. Two types of patterns of seasonal cycles are introduced: input ones $\mathbf{x}$ and output ones $\mathbf{y}$. They are vectors: $\mathbf{x}=[x_1\ x_2 \ldots x_n]^T \in X=\mathbb{R}^n$ and $\mathbf{y}=[y_1\ y_2 \ldots y_n]^T \in Y=\mathbb{R}^n$, with components that are functions of the actual time series points. The x-pattern represents the vector of loads (L) in successive timepoints of the daily period: $\mathbf{L}=[L_1\ L_2 \ldots L_n]^T$. By transforming a time series into a sequence of x-patterns we eliminate a trend and seasonal variations of periods longer than the basic period of length n (daily period in our case; $n=24$ for hourly load time series, $n=48$ for half-hourly load time series, and $n=96$ for quarter-hourly load time series). Moreover, a time series mapped into x-patterns can have some desirable features like stationarity. The goal of y-patterns is to encode and unify the daily cycles to be forecasted using time series characteristics (coding variables) determined from the history. This allows us to determine the forecast of the daily curve having the forecasted y-pattern.

Functions mapping time series points L into x-patterns are dependent on the time series nature (trend, seasonal variations), the forecast period and horizon. They should maximize the model quality. In this work we use the input patterns $\mathbf{x}_i$ (representing the i-th daily period) which components are defined as follows:

$$x_{i,t}=\frac{L_{i,t}-\bar{L}_i}{D_i}, \tag{1}$$

where: $i=1, 2, \ldots, N$ – the daily period number, $t=1, 2, \ldots, n$ – the time series element number in the period i, $L_{i,t}$ – the t-th time series point (load) in the period i, $\bar{L}_i$ – the mean load in the period i, $D_i=\sqrt{\sum_{l=1}^{n}(L_{i,l}-\bar{L}_i)^2}$ – the dispersion of the time series elements in the period i.

Definition (1) expresses normalization of the load vectors $\mathbf{L}_i$ representing the i-th daily period. After normalization we get vectors $\mathbf{x}_i$ having unity length, zero mean and the same variance. Note that the time series which is nonstationary in mean and variance is represented now by x-patterns having the same mean and variance. The trend and additional seasonal variations, i.e. the weekly and annual ones, are filtered. X-patterns carry information only about the shapes of the daily curves.

Whilst x-patterns represent input variables, i.e. loads for the day i, y-patterns represent output variables, i.e. forecasted loads for the day $i+\tau$, where τ is a forecast horizon in days. The output pattern $\mathbf{y}_i$, representing the load vector $\mathbf{L}_{i+\tau}=[L_{i+\tau,1}\ L_{i+\tau,2}\ \ldots\ L_{i+\tau,n}]^T$, has components defined as follows:

$$y_{i,t}=\frac{L_{i+\tau,t}-\bar{L}_i}{D_i}. \tag{2}$$

where: $i=1, 2, \ldots, N$, $t=1, 2, \ldots, n$.

This transformation is similar to (1) but instead of using coding variables $\bar{L}_{i+\tau}$ and $D_{i+\tau}=\sqrt{\sum_{l=1}^{n}(L_{i+\tau,l}-\bar{L}_{i+\tau})^2}$ determined for the day $i+\tau$, in (2) we use coding variables $\bar{L}_i$ and D_i determined for the day i. This is because the coding values for the day $i+\tau$ are not known in the moment of forecasting. Using the known coding values for the day i enables us to calculate the forecast of vector

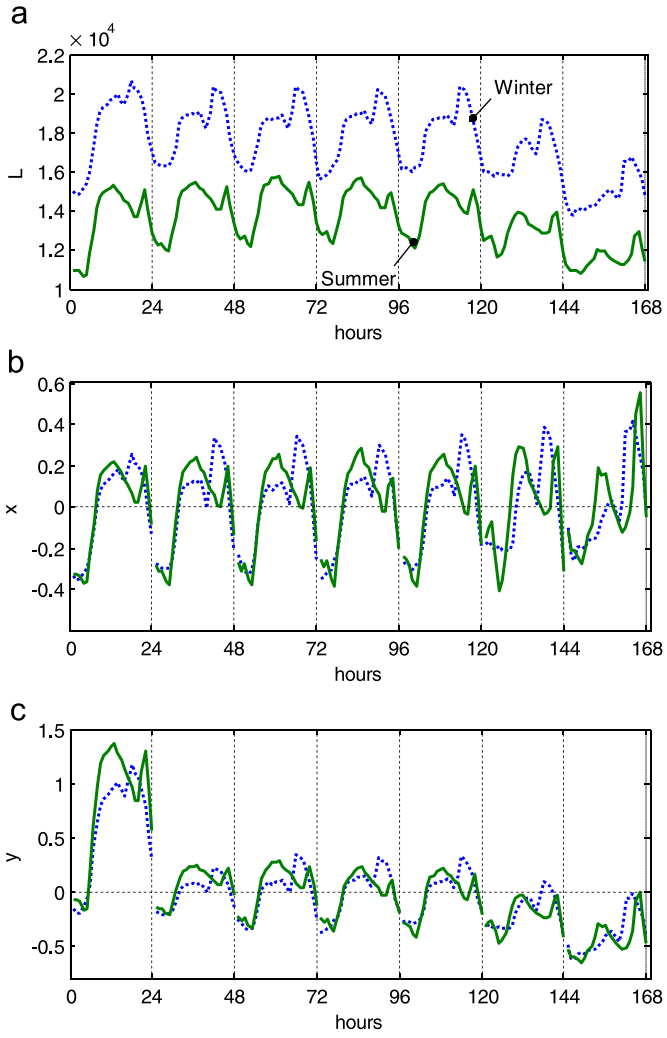


Fig. 2. Two fragments of the load time series (a) and their *x*-patterns (b) and *y*-patterns (c).

$\mathbf{L}_{i+\tau}$ when the forecast of pattern $\mathbf{y}_i$ is generated by the model. To do this we use transformed Eq. (2):

$$\widehat{L}_{i+\tau,t} = \widehat{y}_{i,t} D_i + \overline{L}_i, \tag{3}$$

where $\widehat{y}_{i,t}$ is the forecasted t-th component of the pattern $\mathbf{y}_i$.

Using transformation (1) we unify the level and dispersion of *x*-patterns by subtracting $\overline{L}_i$ (level of the i-th daily period) and dividing the result by D_i (dispersion of the i-th daily period). This is shown in Fig. 2(a) and (b). It can be seen from these figures that the daily cycles of different days of the week $\delta \in$ {Monday,…, Sunday} and from different seasons of the year are represented by *x*-patterns having the same mean and variance.

Using transformation (2) we unify the daily curves in *y*-patterns for each day type δ separately. This is because in (2) we use the coding variables determined for the i-th day to encode $\mathbf{L}_{i+\tau}$. The *y*-patterns of Mondays for $\tau=1$ are located higher than *y*-patterns of other days because we use $\overline{L}_i$ of Sundays in (2), which are usually lower than $\overline{L}_{i+\tau}$ of Mondays. For analogous reasons *y*-patterns of Saturdays and Sundays are located at a lower level than *y*-patterns of weekdays. This is shown in Fig. 2(c). The dispersions of the *y*-patterns are also dependent on the day type δ because in (2) we use coding variables D_i instead of $D_{i+\tau}$.

Because the level and dispersion of *y*-pattern depends on the day type δ, we construct the forecasting model for the particular day of the week using training set containing patterns from the history corresponding to this day type: $\Phi=\{\langle \mathbf{x}_i, \mathbf{y}_i\rangle: i \in \Delta\}$, where Δ is the set of numbers of *y*-patterns representing the same day type δ as the forecasted *y*-pattern. For example when we build the model for Monday and for $\tau=1$ (next day forecast) the training set contains *x*-patterns of Sundays and *y*-patterns of corresponding Mondays from the history. If $\tau=2$ (two days ahead forecast) we use *x*-patterns of Saturdays and *y*-patterns of Mondays in the training set. This approach and the unification of input and output variables using patterns simplify the forecasting model in which we do not need to implement weekly and annual cycles. The information about the position of the daily period in the weekly and annual cycles, which is contained in $\overline{L}_i$, we introduce to the forecast $\widehat{L}_{i+\tau,t}$ in (3) by adding $\overline{L}_i$, as well as we introduce the information about current dispersion of the time series multiplying $\widehat{y}_{i,t}$ by D_i. (It is assumed here that there are strong correlation between levels $\overline{L}_i$ and $\overline{L}_{i+\tau}$, as well as between dispersions D_i and $D_{i+\tau}$.) So when we forecast the load time series using the pattern-based approach, first we filter out the information about dispersion and the position of the days i and $i+\tau$ in the weekly and annual cycles ((1) and (2)). Then we construct the model on patterns and we generate the forecast of the *y*-pattern. Finally we introduce the information about the position of the forecasted day in the weekly and annual cycles and current dispersion using (3).

More functions defining patterns can be found in [21]. A fragment of time series represented by *x*-patterns does not have to coincide with the daily period. It can include several cycles or a part of one cycle. Moreover, it does not have to include contiguous sequence of timepoints. We can select points to the input pattern. We can also use the feature extraction methods to create new pattern components from the original time series (see for example [21] were principal component analysis and partial least-squares regression are used for defining *x*-patterns).

3. Neural models for STLF

In this section we present several approaches to STLF based on NNs: multilayer perceptron (MLP), radial basis function NN (RBFNN), generalized regression NN (GRNN), counterpropagation NN (CPNN), and self organizing map (SOM). A common feature of these approaches is that they work on patterns defined in Section 2. GRNN, CPNN and SOM generate the forecast of $\mathbf{y}$ vector as output, whilst MLP and RBFNN generate the forecast of one component of $\mathbf{y}$ as output. The NNs are optimized and learned separately for each query pattern $\mathbf{x}^*$.

The proposed NN models for pattern-based STLF can be summarized in the following steps:

1. Map the original time series elements to patterns $\mathbf{x}$ and $\mathbf{y}$ using (1) and (2).
2. Select the l nearest neighbors of the query pattern $\mathbf{x}^*$ in Φ and create the set $\Theta=\{\langle \mathbf{x}_i, \mathbf{y}_i\rangle: i \in \Xi\}$, where Ξ is the set of numbers of the l nearest neighbors of $\mathbf{x}^*$.
3. Optimize the NN model in the local leave-one-out procedure.
4. Learn the NN model.
5. Calculate the forecast of $\mathbf{y}$ or y_t for $\mathbf{x}^*$ using NN.
6. Decode the forecast of $\mathbf{y}$ or y_t to get the forecast of load using (3).

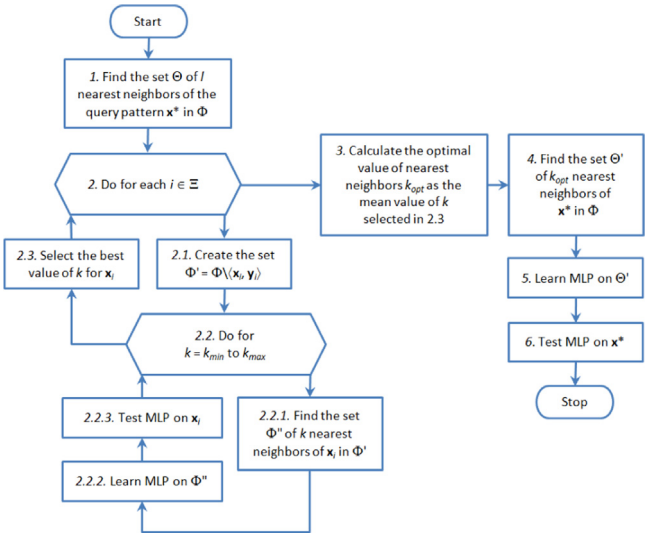
In step 2 the set of l nearest neighbors is selected using Euclidean distance. The dot product can be used instead as a distance measure because vectors $\mathbf{x}$ are normalized. The sets Φ and Θ contain only historical patterns representing the same day types δ as $\mathbf{x}^*$ and corresponding $\mathbf{y}$ which is forecasted.

In step 3 we search for optimal values of hyper-parameters such as number of nearest neighbors k or number of neurons. Because the number of hyper-parameters in the proposed models is small, one or two, the simple grid search is applied to deal with this issue. Optimization is performed in the local leave-one-out cross validation (LLOO), where the validation samples are chosen one by one from the set Θ of nearest neighbors of the query pattern. So we optimize the NN model to get the best performance for the neighborhood of the current query pattern. The learning algorithms applied in step 4 are dependent on the NN type and are described in the next sections.

3.1. Multilayer perceptron

The MLP learns using the local learning procedure [23]. In this case the training samples are selected from the neighborhood of the current query point $\mathbf{x}^*$. By the neighborhood of the query pattern we mean the set of its k nearest neighbors in the training set Φ. The model is optimized and learned to fit accurately to the target function in the neighborhood of $\mathbf{x}^*$ and is competent only for this query pattern. Note that the local complexity of the target function is lower than the global one. So we can use a simple MLP model with not many hidden neurons that learns quickly. The research reported in [23] showed that using only one neuron with sigmoid activation function brings not worse results than networks with several neurons in the hidden layer. So in this work we use one sigmoid neuron as an optimal MLP architecture.

The number of nearest neighbors k (learning points) is the only hyper-parameter tuned in LLOO procedure. The MLP learns using Levenberg–Marquardt algorithm with Bayesian regularization [24], which minimizes a combination of squared errors and net weights. This prevent overfitting. The MLP optimization is summarized in Algorithm 1.

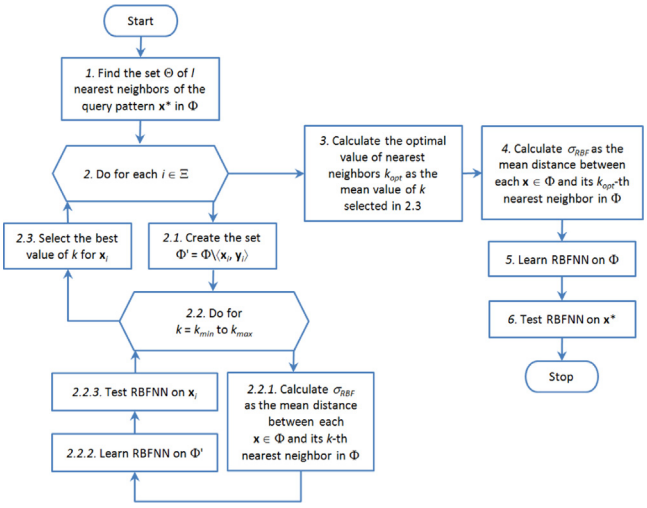


Algorithm 1. MLP forecasting model optimization.

3.2. RBF network

RBFNN learns on the training set Φ. The number of RBF neurons is selected in the learning procedure by adding neurons one by one until the specified error goal is met. Initially the RBF layer has no neurons. A new neuron is added with weights equal to the input pattern $\mathbf{x}$ with the greatest error. (This is implemented in `newrb` function from Neural Networks Toolbox, Matlab 2012b.) The spread of RBFs, σ_{RBF}, determines the smoothness of the fitted function. The larger the spread is, the smoother the function will be. It is assumed that the spread is the same for all neurons and is equal to the mean distance between each $\mathbf{x} \in \Phi$ and its k-th nearest neighbor in Φ. The number k is selected in LLOO procedure. The weights of the linear neurons in the output layer are calculated using simple linear algebra.

Although the RBFNN builds the global model for Φ, it is optimized locally in LLOO. So we can expect good results only for the current query pattern $\mathbf{x}^*$. The RBFNN optimization procedure is shown in Algorithm 2.



Algorithm 2. RBFNN forecasting model optimization.

3.3. Generalized regression neural network

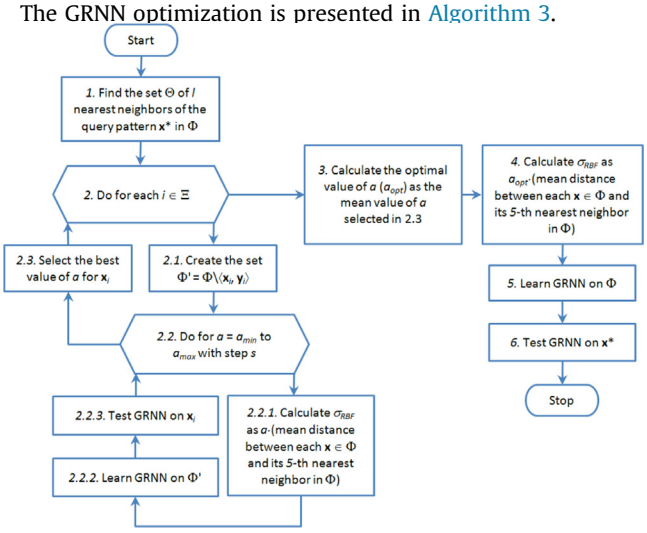
GRNN is a type of RBFNN with a one pass learning and highly parallel structure [25]. This is a memory-based network, where each learning pattern $\mathbf{x}_i$ is represented by one neuron with RBF: $G_i(\mathbf{x})$ with the center $\mathbf{C}_i = \mathbf{x}_i$, $i \in \Delta$. The algorithm provides 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 layer of RBF neurons maps n-dimensional x-pattern space into N_Δ-dimensional space of similarities between input pattern and the training x-patterns: $X \rightarrow [G_1(X)\ G_2(X)\ \ldots\ G_{N\Delta}\ (X)]$, where $N_\Delta = |\Delta|$ is the number of training samples. The output of GRNN is:

$$\widehat{\mathbf{y}} = \frac{\sum_{i=1}^{N_\Delta} G_i(\mathbf{x})\mathbf{y}_i}{\sum_{i=1}^{N_\Delta} G_i(\mathbf{x})}. \tag{4}$$

The only parameter is the RBF spread, σ_{RBF}. As in RBFNN it determines the smoothness of the fitted function. We assume the same spread for all neurons calculated as $\sigma_{RBF} = a \cdot$(the mean distance between each $\mathbf{x} \in \Phi$ and its 5-th nearest neighbor in Φ). The value of a is adjusted in LLOO procedure. In [26] differential evolution was applied for searching for the best values of the N_Δ spreads, i.e. spreads were adjusted individually for each neuron. But this did not bring an expected reduction in the test error.

The GRNN optimization is presented in Algorithm 3.



Algorithm 3. GRNN forecasting model optimization.

3.4. Fuzzy counterpropagation neural network

CPNN in a forward-only version [27] approximates a continuous function by a piecewise constant function (step function). All input points belonging to the same cluster represented by the j-th neuron from the competitive layer are mapped to the same output point stored in the weights coming out of that neuron. To make the fitting function continuous we introduce fuzzy membership the input point $\mathbf{x}$ in clusters represented by neurons.

Two models for STLF based on fuzzy counterpropagation NN (FCPNN) are proposed. In the first one (FCPNN1) weights of the competitive layer neurons are learned using fuzzy c-means clustering [28]. The n-dimensional input space X is partitioned into c fuzzy clusters with centers (or prototypes):

$$\mathbf{w}_j = \frac{\sum_{i=1}^{N_\Delta} \mu(\mathbf{x}_i, \mathbf{w}_j)^m \mathbf{x}_i}{\sum_{i=1}^{N_\Delta} \mu(\mathbf{x}_i, \mathbf{w}_j)^m}, \tag{5}$$

where: $j = 1, 2, \ldots, c$, $m \geq 1$ is the weighting exponent and $\mu(\mathbf{x}_i, \mathbf{w}_j)$ is the degree of membership of $\mathbf{x}_i$ in the j-th cluster.

The membership function $\mu(\mathbf{x}_i, \mathbf{w}_j)$ is defined as [28]:

$$\mu(\mathbf{x}_i, \mathbf{w}_j) = \left(\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_i - \mathbf{w}_j\|}{\|\mathbf{x}_i - \mathbf{w}_k\|} \right)^{\frac{2}{m-1}} \right)^{-1}, \tag{6}$$

where $\|\mathbf{x} - \mathbf{w}\|$ is the distance between pattern $\mathbf{x}$ and a fuzzy center $\mathbf{w}$.

Clusters are represented by hidden neurons and the centers $\mathbf{w}_j$ are the weight vectors of neurons (instar weight vectors). The n-dimensional outstar weight vector $\mathbf{v}_j$ coming out of the j-th neuron is calculated as the mean of y-patterns weighted by the degree of membership of the x-patterns paired with them in the j-th cluster:

$$\mathbf{v}_j = \frac{\sum_{i=1}^{N_\Delta} \mu(\mathbf{x}_i, \mathbf{w}_j)^m \mathbf{y}_i}{\sum_{i=1}^{N_\Delta} \mu(\mathbf{x}_i, \mathbf{w}_j)^m}. \tag{7}$$

The forecast for $\mathbf{x}^*$ is calculated as follows:

$$\widehat{\mathbf{y}} = \frac{\sum_{j=1}^{c} \mu(\mathbf{x}^*, \mathbf{w}_j)^m \mathbf{v}_j}{\sum_{j=1}^{c} \mu(\mathbf{x}^*, \mathbf{w}_j)^m}. \tag{8}$$

Thus the forecast is the mean of the weight vectors $\mathbf{v}_j$ weighted by the degrees of membership of the query pattern $\mathbf{x}^*$ in the clusters $j = 1, 2, \ldots, c$.

In the second approach (FCPNN2) the clusters are created in the space Y by grouping y-patterns using fuzzy c-means method. The cluster centers $\mathbf{v}_j$, which are the n-dimensional outstar weight vectors coming out of the hidden neurons, are:

$$\mathbf{v}_j = \frac{\sum_{i=1}^{N_\Delta} \mu(\mathbf{y}_i, \mathbf{v}_j)^m \mathbf{y}_i}{\sum_{i=1}^{N_\Delta} \mu(\mathbf{y}_i, \mathbf{v}_j)^m}, \tag{9}$$

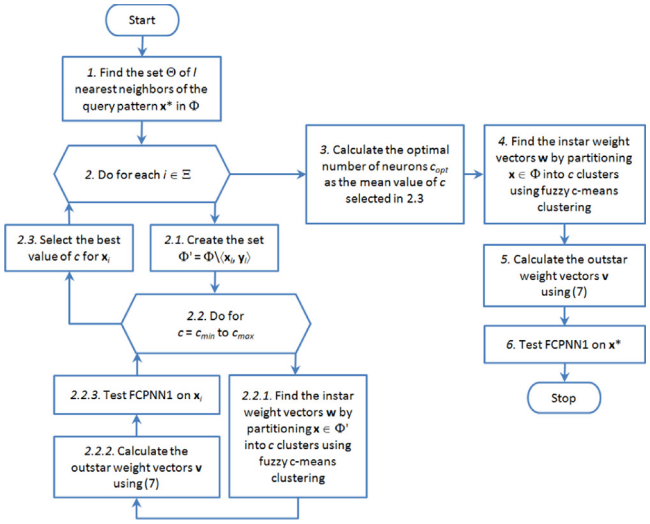
where the membership function $\mu(\mathbf{y}_i, \mathbf{v}_j)$ is defined similarly to (6).

Every of c instar weight vectors $\mathbf{w}_j$ is calculated as the mean of x-patterns weighted by the degree of membership of the y-patterns paired with them in the j-th y-cluster:

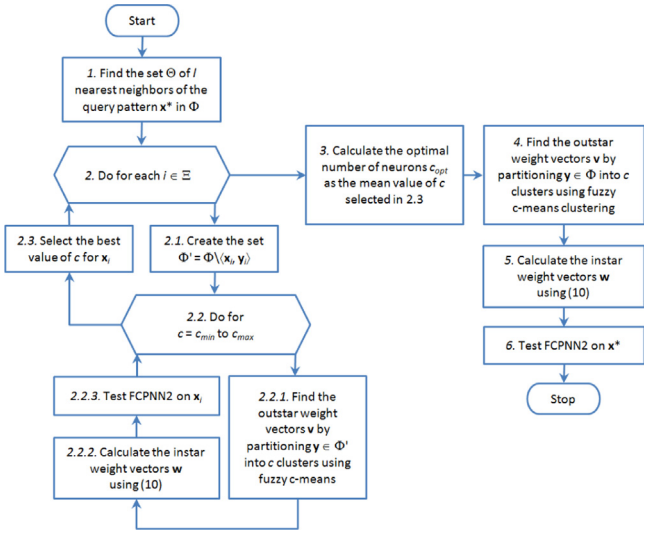
$$\mathbf{w}_j = \frac{\sum_{i=1}^{N_\Delta} \mu(\mathbf{y}_i, \mathbf{v}_j)^m \mathbf{x}_i}{\sum_{i=1}^{N_\Delta} \mu(\mathbf{y}_i, \mathbf{v}_j)^m}. \tag{10}$$

The forecast $\widehat{\mathbf{y}}$ is determined in the same way as in the first approach, according to (8). The only difference between FCPNN1 and FCPNN2 is the way of weight vectors $\mathbf{w}_j$ and $\mathbf{v}_j$ learning. In FCPNN1 the instar weight vectors $\mathbf{w}_j$ represent centers of the x-pattern clusters. And the outstar weight vectors $\mathbf{v}_j$ represent mean of y-patterns weighted by the membership of the x-patterns paired with them in the j-th cluster. Note that the y-patterns represented by $\mathbf{v}_j$ can be dispersed in the Y-space and do not form a compact clusters (i.e. the clusters overlap). In FCPNN2 the clusters are detected in Y-space and represented by the outstar weight vectors $\mathbf{v}_j$. So they represent neighboring y-patterns. But now the instar weight vectors $\mathbf{w}_j$ can represent dispersed x-patterns which do not form compact clusters in X-space.

The two versions of FCPNN described above have the same architecture shown in Fig. 3. The optimization procedures for FCPNN1 and FCPNN2 in which we search for the optimal number of neurons using LLOO in Algorithms 4 and 5 are shown.



Algorithm 4. FCPNN1 forecasting model optimization.



Algorithm 5. FCPNN2 forecasting model optimization.

3.5. Self organizing map

Three approaches are proposed for STLF based on SOM. In the first one (SOM1) the paired x- and y-patterns are concatenated and form $2n$-dimensional vectors $\mathbf{z}_i=[\mathbf{x}_i^T\ \mathbf{y}_i^T]^T$, $i\in\Delta$. SOM learns on vectors $\mathbf{z}_i$ using "winner takes most" algorithm. After learning each instar weight vector of the j-th neuron ($\mathbf{w}_j$) represents the cluster center. The weights vectors have two n-dimensional parts corresponding to x- and y-patterns: $\mathbf{w}_j=[\mathbf{w}_{x,j}^T\ \mathbf{w}_{y,j}^T]^T$, $j=1, 2, \ldots, c$. In the forecasting phase the query pattern $\mathbf{x}^*$ is presented and the winning neuron is detected as that one having the nearest $\mathbf{w}_x$ to $\mathbf{x}^*$:

$$j^*=\arg\min_{j=1,2,\ldots,c}\|\mathbf{x}^*-\mathbf{w}_{x,j}\|. \tag{11}$$

The y-part of weight vector of the winner is the forecast for $\mathbf{x}^*$:

$$\widehat{\mathbf{y}}=\mathbf{w}_{y,j^*}. \tag{12}$$

Thus the forecasted y-pattern is the mean of y-patterns forming the nearest cluster represented by j^*-th neuron. The number of neurons c is selected in LLOO procedure (see Algorithm 6). Increasing c we increase the model variance and decrease its bias (more accurate fitting to training data).

In the second approach (SOM2) inspired by [29] patterns $\mathbf{x}$ and $\mathbf{y}$ from Φ are clustered independently by two SOMs. After learning the weight vectors of the first SOM ($\mathbf{w}_{x,j}$, $j=1, 2, \ldots, c_x$) represent centers of the x-pattern clusters, while the weight vectors of the second SOM ($\mathbf{w}_{y,k}$, $k=1, 2, \ldots, c_y$) represent centers of the y-pattern clusters. On the basis of the training set the conditional probabilities $P(n_{y,k}|n_{x,j})$ that the pattern $\mathbf{y}_i$ belongs to the cluster represented by the k-th neuron ($n_{y,k}$) of the second SOM (SOMy), when the corresponding pattern $\mathbf{x}_i$ belongs to the cluster represented by the j-th neuron ($n_{x,j}$) of the first SOM (SOMx) are estimated. (This is done by presenting the successive training pairs $\langle\mathbf{x}_i, \mathbf{y}_i\rangle$ to both SOMs and counting the number of wins of each pair of neurons $n_{y,k}$ and $n_{x,j}$.)

In the forecasting phase the query pattern $\mathbf{x}^*$ is presented to SOMx and the j^*-th neuron is selected as the winner. The forecasted y-pattern is calculated as the mean of the weight vectors of SOMy $\mathbf{w}_{y,k}$ weighted by the conditional probabilities $P(n_{y,k}|n_{x,j^*})$:

$$\widehat{\mathbf{y}}=\frac{\sum_{k=1}^{c_y}P(n_{y,k}|n_{x,j^*})\mathbf{w}_{y,k}}{\sum_{k=1}^{c_y}P(n_{y,k}|n_{x,j^*})}. \tag{13}$$

The weight vectors of these SOMy neurons, which probability of winning is the highest after having observed the winning of the j^*-th SOMx neuron, have the largest share in mean (13). The numbers of neurons c_x and c_y were selected using grid search in LLOO procedure according to Algorithm 7.

In the third approach (SOM3) the SOM learns using only y-patterns. The training set contains all y-patterns from the history: $\Omega=\{\mathbf{y}_i:\ i=1, 2, \ldots, N\}$. After learning n-dimensional weight vectors $\mathbf{w}_j$, $j=1, 2, \ldots, c$ represent centers of the y-pattern clusters. The neurons are labeled. The label contains information about days represented by y-patterns belonging to the cluster: the day numbers i and the day types δ. For example the label of the neuron representing five y-patterns can be in the form: ⟨398 – Tue, 399 – Wed, 400 – Thu, 764 – Thu, 765 – Wed⟩, i.e. five entries: day number–day type.

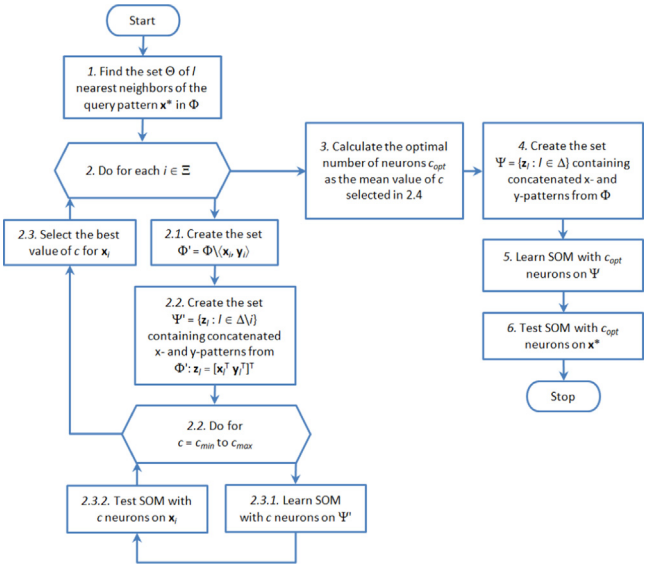
To forecast the y-pattern having the number i^* and representing the day type δ^* all labels are searched. For each label we count the number of entries e_j which satisfy two conditions:

1. the day is from the same period of the year as the forecasted y-pattern:
$$i\in[i^*-L, i^*-1]\vee[i^*-A-L, i^*-A+L]\vee[i^*-n\cdot A-L, i^*-n\cdot A+L], \tag{14}$$
where L is the number of days defining the period length, e.g. 30, A is the annual period (365 or 366 days), and $n=1, 2, \ldots$ is dependent on the length of the time series.
2. the day type is the same as for the forecasted y-pattern: $\delta=\delta^*$.

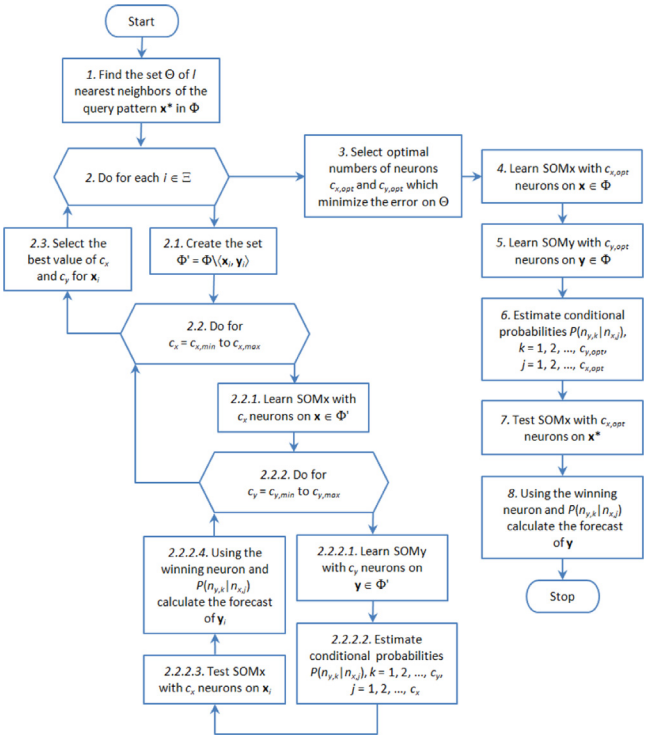
The forecasted y-pattern is calculated as the mean of the weight vectors $\mathbf{w}_j$ weighted by the numbers of the corresponding label entries satisfying the above conditions:

$$\widehat{\mathbf{y}}=\frac{\sum_{j=1}^{c}e_j\mathbf{w}_j}{\sum_{j=1}^{c}e_j} \tag{15}$$

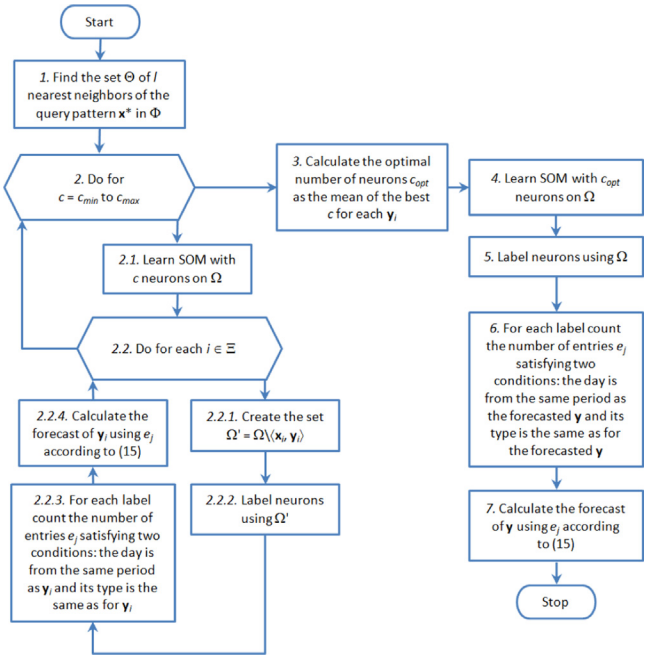
The number of neurons c is selected in LLOO procedure as shown in Algorithm 8.



Algorithm 6. SOM1 forecasting model optimization.



Algorithm 7. SOM2 forecasting model optimization.



Algorithm 8. SOM3 forecasting model optimization.

4. Simulation study

In this section the proposed neural models are tested on STLF problems (one day ahead forecasts, $\tau=1$) using four real load time series:

- PL: time series of the hourly load of the Polish power system from the period of 2002–2004. The test sample includes data from 2004 with the exception of 13 atypical days (e.g. public holidays),
- FR: time series of the half-hourly load of the French power system from the period of 2007–2009. The test sample includes data from 2009 except for 21 atypical days,
- GB: time series of the half-hourly load of the British power system from the period of 2007–2009. The test sample includes data from 2009 except for 18 atypical days,
- VC: time series of the half-hourly load of the power system of Victoria, Australia, from the period of 2006–2008. The test sample includes data from 2008 except for 12 atypical days.

The number of nearest neighbors in LLOO procedures in all cases was set to $l=12$. Other settings are:

- for MLP (see Algorithm 1): $k_{min}=2$, $k_{max}=30$,
- for RBFNN (see Algorithm 2): $k_{min}=2$, $k_{max}=20$,
- for GRNN (see Algorithm 3): $a_{min}=0.2$, $a_{max}=1.6$, $s=0.2$,
- for FCPNN1 (see Algorithm 4): $c_{min}=2$, $c_{max}=50$,
- for FCPNN2 (see Algorithm 5): $c_{min}=2$, $c_{max}=50$,
- for SOM1 (see Algorithm 6): $c_{min}=2$, $c_{max}=50$,
- for SOM2 (see Algorithm 7): $c_{x,min}=2$, $c_{x,max}=50$, $c_{y,min}=2$, $c_{y,max}=50$,
- for SOM3 (see Algorithm 8): $L=30$, $c_{min}=50$, $c_{max}=150$.

The neural models were compared with ARIMA and exponential smoothing (ES). In ARIMA and ES we decompose the time series into n separate series: one series for each $t=1, 2, \ldots, n$. This eliminates the daily seasonality and simplifies the forecasting problem. The ARIMA and ES parameters were estimated for each forecasting task (i.e. the forecast of system load at time t of the day $i+\tau$) using 12-week time series fragments immediately preceding the forecasted day. Atypical days in these fragments were replaced with the days from the previous weeks. Due to using short time series fragments for parameter estimation (much shorter than the annual period) and due to time series decomposition into n series we do not have to take into account the annual and daily seasonalities in the models. In such case the number of parameters is much smaller and they are easier to estimate compared to models with triple seasonality.

For each forecasting task we create seasonal ARIMA$(p, d, q)\times(P, D, Q)_v$ model (where the period of the seasonal pattern appearing $v=7$, i.e. one week period) as well as the ES state space model. ES models are classified into 30 types [30] depending on how seasonal, trend and error components are taken into account (they can be expressed additively or multiplicatively, and the trend can be damped or not). To estimate parameters of ARIMA and ES the stepwise procedures for traversing the model spaces implemented in the `forecast` package for the R environment for statistical computing [31] were used. These automatic procedures return the optimal models with the lowest Akaike information criterion value.

The forecasting errors for the test samples (mean absolute percentage errors which are traditionally used in STLF, MAPE$=100\cdot$mean(|(forecast − actual value)/actual value|)) and their interquartile ranges as a measure of error dispersion in Table 1 are shown. In this table the errors of the naïve method of the form: the forecasted daily curve is the same as seven days ago, are also presented for comparison. The errors generated by the models were compared in pairs and the Wilcoxon rank sum test with 5% significance level was used to confirm statistically significant difference between them. The statistically best results are marked in Table 1 with an asterisk and the second best results are marked with a double asterisk.

As we can see from this table GRNN takes the first place among tested models for all data. MLP was not much worse. In Fig. 4 the ranking of the models are presented based on the mean error on

the test sets for all time series. The worse among neural models turned out to be those based on SOM. The conventional forecasting models: ARIMA and ES work significantly worse than the best NN models, but ES was better than ARIMA in all cases. Note that the ARIMA and ES are optimized on the time series fragments (12-week fragment in our case) directly preceding the forecasted fragment. In the proposed approach the neural models are optimized on the selected patterns which are the most similar to the query pattern. These patterns represent fragments of the load time series from different periods.

More detailed results, the probability density functions (PDF) of the percentage errors (PE = 100 · (forecast − actual value)/actual value) estimated using kernel density estimation, are shown in Fig. 5. Note that only small differences can be observed in the PDF for MLP and GRNN.

The learning and optimization procedure of the winning neural STLF model, GRNN, is the simplest among other models. Note that in GRNN only one parameter is estimated, the spread. The dimensions of input and output patterns do not affect the number of parameters to estimate unlike in other tested models. This should be considered as a valuable property. The second winner, one-neuron perceptron, has $n+1$ weights to learn and the number of nearest neighbors k to select for each timepoint of the forecasted period t. FCPNN and SOM models require data clustering and more complex learning. It is worth noting that the GRNN model is very similar to the STLF model based on Nadaraya–Watson estimator proposed in [32]. The only difference is that we use the product kernels as RBFs in the Nadaraya–Watson model. The spreads (or bandwidths) in the case of product kernels are determined for each dimension. So instead of one parameter as in the case of GRNN we estimate n parameters. Errors achieved by both models are similar (see [26]).

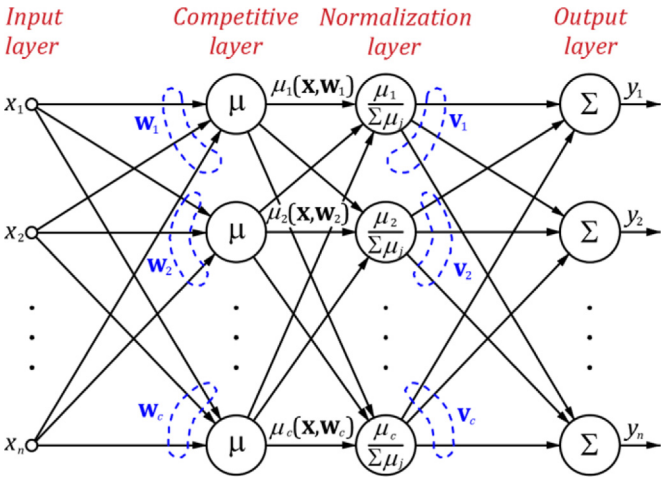


Fig. 3. The FCPNN architecture.

Table 1
Forecast errors and their interquartile ranges for the proposed and benchmark models.

Model	PL		FR		GB		VC	
	MAPE	*IQR*	*MAPE*	*IQR*	*MAPE*	*IQR*	*MAPE*	*IQR*
MLP	**1.45****	1.38	**1.59***	1.64	**1.63****	1.68	**2.99****	2.74
RBFNN	1.67	1.53	**1.70****	1.70	1.84	1.90	3.23	3.05
GRNN	**1.38***	1.33	**1.64***	1.71	**1.56***	1.64	**2.83***	2.59
FCPNN1	1.71	1.46	1.90	1.95	1.69	1.79	3.18	2.97
FCPNN2	1.63	1.50	1.82	1.86	1.66	1.71	3.22	2.99
SOM1	1.74	1.65	2.10	2.19	1.95	1.98	3.41	3.12
SOM2	1.73	1.53	1.95	2.04	1.78	1.89	3.28	3.08
SOM3	1.99	1.83	2.06	2.18	1.95	2.02	3.63	3.47
ARIMA	1.82	1.71	2.32	2.53	2.02	2.07	3.67	3.42
ES	1.66	1.57	2.10	2.29	1.85	1.84	3.52	3.35
Naïve	3.43	3.42	5.05	5.96	3.52	3.82	4.54	4.20

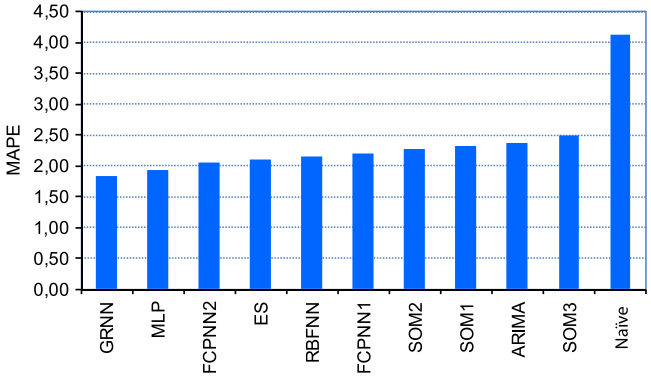


Fig. 4. Ranking of the STLF models.

Patterns of the time series sequences enable us to simplify the problem of forecasting non-stationary time series with multiple seasonal cycles and trend. Time series representation by patterns can be used in various forecasting models. In [22] local linear regression models based on patterns were proposed. The relationship between input and output patterns is modeled locally in the neighborhood of query pattern using linear regression. Results generated by the best linear models, i.e. principal components regression and partial least-squares regression are comparable with results generated by the best NN models (see Table 2). In these both linear models the components of input patterns are constructed by linear combination of original components.

Another group of STLF models using patterns: models based on the similarity between patterns of seasonal cycles are presented in [33]. They include: Nadaraya–Watson estimator, nearest neighbor estimation-based models and pattern clustering-based models such as classical clustering methods and new artificial immune systems. These models construct the regression curve aggregating the forecast patterns from the history with weights dependent on the similarity between input patterns paired with the forecast patterns. Simulation studies reported in [33] were performed on the same datasets as in this work so you can compare results (see Table XI in [33]). Table 2 summarizes results for the above mentioned models.

5. Conclusions

The major contribution of this work is to propose and compare new neural STLF models which learn using patterns representing daily load curves. Due to patterns the problem of forecasting multiple seasonal nonstationary time series simplifies. The seasonal variations of periods longer than the daily one as well as a trend observed in load time series are filtered. Thus the forecasting model does not have to capture the complex nature of the process expressed in the time series.

The forecasting results demonstrate that the neural models learned using patterns perform remarkably well. The model based on GRNN turned out to be the most accurate in STLF compared to other neural models: MLP, RBFNN, FCPNNs and SOMs, as well as the classical statistical models: ARIMA and ES. GRNN is the simplest among tested methods. It has only one parameter to estimate, the spread of RBFs. Such a model is easy to optimize and has good generalization properties. Its learning and optimization procedures are extremely fast. Another valuable feature of GRNN is

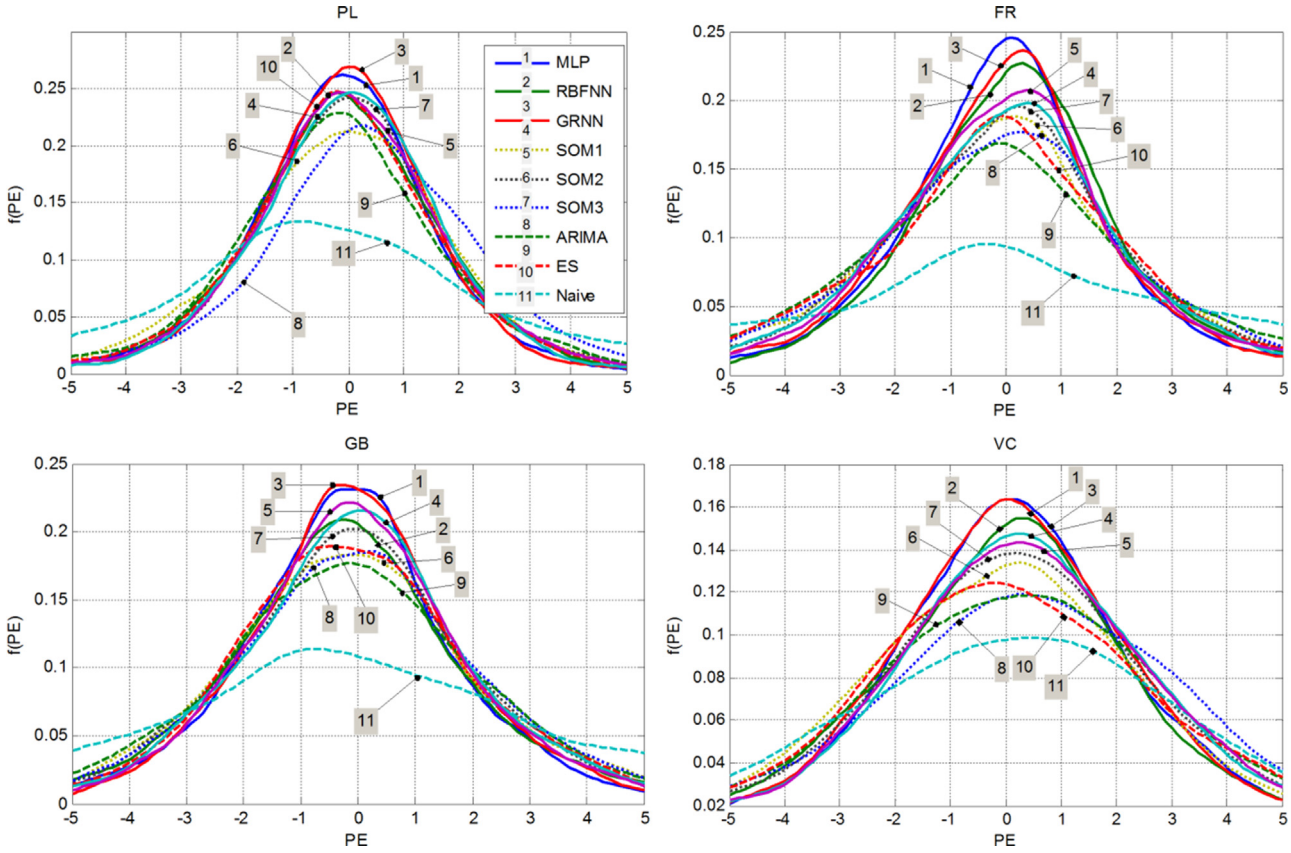


Fig. 5. Probability distribution functions of the percentage errors (PEs).

Table 2
Forecast errors and their interquartile ranges for state-of-the-art pattern based STLF models.

Model	PL		FR		GB		VC	
	MAPE	*IQR*	*MAPE*	*IQR*	*MAPE*	*IQR*	*MAPE*	*IQR*
PCR	1.35	1.33	1.71	1.78	1.60	1.68	3.00	2.70
PLSR	1.34	1.32	1.57	1.61	1.54	1.61	2.83	2.60
N-WE	1.30	1.30	1.66	1.67	1.55	1.63	2.82	2.56
FNM	1.38	1.38	1.67	1.71	1.60	1.66	2.91	2.67
FP1+*k*-means	1.69	1.64	2.05	2.17	1.84	1.88	3.34	3.01
FP2+*k*-means	1.59	1.51	1.94	2.05	1.76	1.84	3.13	2.94
AIS1	1.50	1.50	1.93	1.95	1.77	1.84	3.04	2.75
AIS2	1.50	1.51	1.93	1.96	1.78	1.87	3.33	2.93

where:
PCR – principal components regression [22]
PLSR – partial least-squares regression [22]
N-WE – Nadaraya–Watson estimator [33]
FNM – fuzzy neighborhood model [33]
FP1+*k*-means – model based on *k*-means clustering of concatenated *x*- and *y*-patterns (similarly to SOM1) [33]
FP2+*k*-means – model based on *k*-means clustering of *x*- and *y*-patterns independently (similarly to SOM2) [33]
AIS1 – artificial immune system working on concatenated *x*- and *y*-patterns (similarly to SOM1) [33]
AIS2 – artificial immune system working on separate populations of patterns: type *x* and *y* (similarly to SOM2) [33].

its ability to predict all components of *y*-pattern at once. The dimension of *y*-pattern as well as the dimension of *x*-pattern do not affect the number of parameters to estimate. The second best model, MLP, predict only one component of *y*-pattern at once. Due to the local approach to MLP learning only one-neuron model is sufficient to approximate the target function in the neighborhood of the query pattern.

The forecasting accuracy of the neural models was increased due to the local approach to the model optimization. The model hyper-parameters were tuned in the local version of leave-one-out to get the best performance in the neighborhood of the query pattern.

In general, the proposed approaches can be applied for different power systems. The future improvement may include taking into account additional input variables such as weather conditions. This can be done by building correction models that learn relationships between errors of the forecasts generated by the basic model and weather factors. Another improvement relates to construction specialized forecasting models for atypical days. In the future work we are going to apply the proposed models to other time series representing processes and phenomena from different areas: economy, meteorology, industry etc.

List of symbols

$\mathbf{x}=[x_1\ x_2\ \ldots\ x_n]^T \in X=\mathbb{R}^n$ – the input pattern
$\mathbf{y}=[y_1\ y_2\ \ldots\ y_n]^T \in Y=\mathbb{R}^n$ – the output pattern
$\mathbf{x}^*$ – the query pattern
$\widehat{\mathbf{y}}$ – the forecast of *y*-pattern
$\mathbf{w}=[w_1\ w_2\ \ldots\ w_n]^T \in X=\mathbb{R}^n$ – the instar weight vector of the neuron from the competitive layer (in SOMs and FCPNNs)
$\mathbf{v}=[v_1\ v_2\ \ldots\ v_n]^T \in Y=\mathbb{R}^n$ – the outstar weight vector of the neuron from the competitive layer in FCPNNs
c – the number of neurons representing clusters in SOMs and FCPNNs
$\delta \in$ {Monday, …, Sunday} – day of the week
Δ – the set of numbers of *y*-patterns from the history representing the same day type δ as the forecasted *y*-pattern
$N_\Delta = |\Delta|$ – the number of training samples in Φ

N – the number of training samples in Ω
n – length of the daily period and the number of components in vectors $\mathbf{x}$ and $\mathbf{y}$
Ξ – the set of numbers of the l nearest neighbors of $\mathbf{x}^*$ in Φ
$\Phi = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle : i \in \Delta\}$ – the training set containing pairs of patterns from the history, such that $\mathbf{y}_i$ represent the same day of the week δ as the forecasted y-pattern
$\Theta = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle : i \in \Xi\}$ – the set containing pairs of patterns, such that $\mathbf{x}_i$ is one of the l nearest neighbors of $\mathbf{x}^*$ in Φ
$\Omega = \{\mathbf{y}_i : i = 1, 2, \ldots, N\}$ – the training set for SOM3 containing y-patterns from the history

Acknowledgment

I am very grateful to James W. Taylor with the Saïd Business School, University of Oxford for supplying British and French data, and Shu Fan and Rob J. Hyndman with the Business and Economic Forecasting Unit, Monash University for supplying Australian data.

This work was supported in part by the Polish Ministry of Science and Higher Education under Grant no. N516 415338.

References

[1] R. Weron, Modeling and Forecasting Electricity Loads and Prices, Wiley, Chichester, 2006.
[2] J.W. Taylor, Short-term load forecasting with exponentially weighted methods, IEEE Trans. Power Syst. 27 (1) (2012) 458–464.
[3] J. Nowicka-Zagrajek, R. Weron, Modeling electricity loads in California: ARMA models with hyperbolic noise, Signal Process. 82 (2002) 1903–1915.
[4] C.-M. Lee, C.-N. Ko, Short-term load forecasting using lifting scheme and ARIMA models, Expert Syst. Appl. 38 (2011) 5902–5911.
[5] S. Tzafestas, E. Tzafestas, Computational intelligence techniques for short-term electric load forecasting, J. Intell. Robot. Syst. 31 (2001) 7–68.
[6] K. Metaxiotis, A. Kagiannas, D. Askounis, J. Psarras, Artificial intelligence in short term electric load forecasting: a state-of-the-art survey for the researcher, Energy Convers. Manag. 44 (2003) 1525–1534.
[7] H.S. Hippert, J.W. Taylor, An evaluation of Bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting, Neural Netw. 23 (2010) 386–395.
[8] N. Amjady, F. Keynia, Short-term load forecasting of power systems by combination of wavelet transform and neuro-evolutionary algorithm, Energy 34 (2009) 46–57.
[9] Z.A. Bashir, M.E. El-Hawary, Applying wavelets to short-term load forecasting using PSO-based neural networks, IEEE Trans. Power Syst. 24 (1) (2009) 20–27.
[10] Y. Chen, P.B. Luh, C. Guan, Y. Zhao, L.D. Michel, M.A. Coolbeth, P.B. Friedland, S. J. Rourke, Short-term load forecasting: similar day-based wavelet neural networks, IEEE Trans. Power Syst. 25 (1) (2010) 322–330.
[11] H. Kebriaei, B.N. Araabi, A. Rahimi-Kian, Short-term load forecasting with a new nonsymmetric penalty function, IEEE Trans. Power Syst. 26 (4) (2011) 1817–1825.
[12] Z. Yun, Z. Quan, S. Caixin, L. Shaolan, L. Yuming, S. Yang, RBF neural network and ANFIS-based short-term load forecasting approach in real-time price environment, IEEE Trans. Power Syst. 23 (3) (2008) 853–858.
[13] S.J. Yao, Y.H. Song, L.Z. Zhang, X.Y. Cheng, Wavelet transform and neural networks for short-term electrical load forecasting, Energy Convers. Manag. 41 (2000) 1975–1988.
[14] O.A.S. Carpinteiro, A.J.R. Reis, A.P. Alves da Silva, A hierarchical neural model in short-term load forecasting, Appl. Soft Comput. 4 (2004) 405–412.
[15] S. Fan, L. Chen, Short-term load forecasting based on an adaptive hybrid method, IEEE Trans. Power Syst. 21 (1) (2006) 392–401.
[16] V. Yadav, D. Srinivasan, A SOM-based hybrid linear-neural model for short-term load forecasting, Neurocomputing 74 (2011) 2874–2885.
[17] A. Lendasse, M. Cottrell, V. Wertz, M. Verleysen, Prediction of electric load using Kohonen maps – application to the Polish electricity consumption, in: Proceedings of the American Control Conference ACC, 2002, pp. 3684–3689.
[18] A. Lendasse, J. Lee, V. Wertz, M. Verleysen, Forecasting electricity consumption using nonlinear projection and self-organizing maps, Neurocomputing 48 (2002) 299–311.
[19] H.S. Hippert, C.E. Pedreira, R.C. Souza, Neural networks for short-term load forecasting: a review and evaluation, IEEE Trans. Power Syst. 16 (1) (2001) 44–55.
[20] V.S. Kodogiannis, E.M. Anagnostakis, Soft computing based techniques for short-term load forecasting, Fuzzy Sets Syst. 128 (2002) 413–426.
[21] G. Dudek, Pattern similarity-based methods for short-term load forecasting – Part 1: principles, Appl. Soft Comput. 37 (2015) 277–287.
[22] G. Dudek, Pattern-based local linear regression models for short-term load forecasting, Electric Power Syst. Res. 130 (2016) 139–147.
[23] G. Dudek, Forecasting time series with multiple seasonal cycles using neural networks with local learning, in: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L.A. Zadeh, J.M. Zurada (Eds.), Artificial Intelligence and Soft Computing, ICAISC 2013, LNCS 7894, 2013, pp. 52–63.
[24] F.D. Foresee, M.T. Hagan, Gauss-Newton approximation to Bayesian regularization, in: Proceedings of Conference Neural Networks, 1997, pp. 1930–1935.
[25] D.F. Specht, A general regression neural network, IEEE Trans. Neural Netw. 2 (6) (1991) 568–576.
[26] G. Dudek, Generalized regression neural network for forecasting time series with multiple seasonal cycles, in: D. Filev, J. Jablkowski, J. Kacprzyk, M. Krawczak, I. Popchev, L. Rutkowski, V. Sgurev, E. Sotirova, P. Szynkarczyk, S. Zadrozny (Eds.), Advances in Intelligent Systems and Computing 323, Intelligent Systems 2015, 2014', pp. 839–846.
[27] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley Publishing Company, 1990.
[28] J.C. Bezdek, Pattern Recogniition with Fuzzy Objective Function Algorithms, Plenum Press, New York and London, 1981.
[29] A. Lendasse, M. Verleysen, E. de Bodt, M. Cottrell, P. Gregoire, Forecasting time-series by Kohonen classification, in: Proceedings of the European Symposium on Artificial Neural Networks, 1998, pp. 221–226.
[30] R.J. Hyndman, A.B. Koehler, J.K. Ord, R.D. Snyder, Forecasting with Exponential Smoothing: The State Space Approach, Springer-Verlag, Berlin, Heidelberg, 2008.
[31] R.J. Hyndman, Y. Khandakar, Automatic time series forecasting: the forecast package for R, J. Stat. Softw. 27 (3) (2008) 1–22.
[32] G. Dudek, Short-term load forecasting based on kernel conditional density estimation, Prz. Elektrotech. (Electr. Rev.) 86 (8) (2010) 164–167.
[33] G. Dudek, Pattern similarity-based methods for short-term load forecasting – Part 2: models, Appl. Soft Comput. 36 (2015) 422–441.

Grzegorz Dudek received his Ph.D. degree in electrical engineering from the Czestochowa University of Technology, Poland, in 2003 and habilitation degree in computer science from Lodz University of Technology, Poland, in 2013. Currently, he is an associate professor at the Department of Electrical Engineering, Czestochowa University of Technology. He is the author of two books concerning load forecasting and power system operation and over 70 scientific papers. His research interests include pattern recognition, machine learning, artificial intelligence, and their application to classification, regression, forecasting and optimization problems especially in power engineering.