A Hybrid Residual Dilated LSTM and Exponential Smoothing Model for Midterm Electric Load Forecasting

Grzegorz Dudek, Paweł Pełka, and Slawek Smyl

***Abstract*— This work presents a hybrid and hierarchical deep learning model for midterm load forecasting. The model combines exponential smoothing (ETS), advanced long short-term memory (LSTM), and ensembling. ETS extracts dynamically the main components of each individual time series and enables the model to learn their representation. Multilayer LSTM is equipped with dilated recurrent skip connections and a spatial shortcut path from lower layers to allow the model to better capture long-term seasonal relationships and ensure more efficient training. A common learning procedure for LSTM and ETS, with a penalized pinball loss, leads to simultaneous optimization of data representation and forecasting performance. In addition, ensembling at three levels ensures a powerful regularization. A simulation study performed on the monthly electricity demand time series for 35 European countries confirmed the high performance of the proposed model and its competitiveness with classical models such as ARIMA and ETS as well as state-of-the-art models based on machine learning.**

***Index Terms*— Deep learning, exponential smoothing, long short-term memory, midterm load forecasting (MTLF), recurrent neural networks (NNs), time series forecasting.**

I. Introduction

ELECTRICITY demand forecasting is an essential tool in all sectors in the electric power industry. Midterm load forecasting (MTLF), which involves forecasting the daily peak load for future months as well as monthly electricity demand, is necessary for power system operation and planning in such areas as maintenance scheduling, fuel reserve planning, hydrothermal coordination, planning of electrical energy import and export, and also security assessment. In deregulated power systems, MTLF is a basis for the negotiation of forward contracts. Forecast accuracy translates directly into financial performance for energy companies and energy market participants. The financial impact can be measured in millions of dollars for every point of forecasting accuracy gained. All the above reasons justify interest in new forecasting tools for MTLF.

Manuscript received March 26, 2020; revised October 4, 2020; accepted December 12, 2020. Date of publication January 8, 2021; date of current version July 7, 2022. *(Corresponding author: Grzegorz Dudek.)*

Grzegorz Dudek and Paweł Pełka are with the Department of Electrical Engineering, Częstochowa University of Technology, 42-200 Częstochowa, Poland (e-mail: dudek@el.pcz.czest.pl; p.pelka@el.pcz.czest.pl).

Slawek Smyl is with Uber Technologies, San Francisco, CA 94104 USA (e-mail: slaweks@hotmail.co.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3046629.

Digital Object Identifier 10.1109/TNNLS.2020.3046629

In this work, we focus on monthly electricity demand forecasting. Monthly electricity demand time series express a nonlinear trend, yearly cycles, and a random component. The trend is dependent on the rate of economic development in a country, while seasonal cycles are related to the climate, weather factors, and the variability of seasons [1]. Factors disrupting electricity demand in a midterm horizon include unpredictable economic events and political decisions [2].

MTLF methods can be divided into statistical/econometric models or machine learning (ML)/computational intelligence models [3]. Typical examples of the former are ARIMA, ETS, and linear regression. ARIMA and ETS can deal with seasonal time series, but linear regression requires additional operations, such as decomposition or the extension of the model with periodic components [4]. Statistical models have undoubted advantages, being relatively simple, robust, efficient, and automatic, so that they can be used by nonexpert users.

The flexibility of ML models has increased researchers' interest in them as MTLF tools [5]. Of these, neural networks (NNs) are the most explored because of their attractive features such as learning capability, universal approximation property, nonlinear modeling, massive parallelism, and ease of specifying a loss function, to align it better with forecasting goals. Some examples of using different architectures of NNs for MLTF are: [6] where NN learns on historical demand and weather factors, [7] where Kohonen NN was used, [8] where NNs were supported by fuzzy logic, [9] where generalized regression NN was used, [10] where NNs, linear regression, and AdaBoost were used, [11] where weighted evolving fuzzy NNs were combined, and [12] where recurrent NNs were used. Among other ML MTLF models, the following can be mentioned: support vector machines [13] and pattern similarity-based models [14].

Recent trends in ML such as deep learning, especially deep recurrent NNs (RNNs), are very attractive for time series forecasting [15]. RNNs with connections between nodes forming a directed graph along a temporal sequence are able to exhibit temporal dynamic behavior using their internal state (memory) to process sequences of inputs. Recent works have reported that RNNs, such as the long short-term memory (LSTM), provide high accuracy in forecasting and outperform most of the traditional statistical and ML methods, such as ARIMA, support vector machine, and shallow NNs [16].

There are many examples of an application of LSTMs to load forecasting: [17]–[19].

Many new ideas in the field of deep learning have been successfully applied to time series forecasting. For example, in [20], bidirectional LSTM is proposed for short-term scheduling in power markets. This solution has two benefits: long-range memory and bidirectional processing. It takes advantage of deep architectures, which are able to build up progressively higher level representations of data, by piling up RNN layers on top of each other. A residual recurrent highway network for learning deep sequence predictions was proposed in [21]. It contains highways within the temporal structure of the network for unimpeded information propagation, thus alleviating gradient vanishing problem. Hierarchical structure learning is posed as a residual learning framework to prevent performance degradation problems. Another example of using a new deep learning solution for time series forecasting is the N-BEATS model proposed is [22]. Its architecture is based on backward and forward residual links and a deep stack of fully connected layers. N-BEATS has a number of desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train.

In addition to point forecasting, deep learning also enables probabilistic forecasting. A model proposed in [23], DeepAR, produces accurate probabilistic forecasts, based on training an autoregressive RNN on a large number of related time series. This model makes probabilistic forecasts in the form of Monte Carlo samples that can be used to compute consistent quantile estimates for all subranges in the prediction horizon. Another solution for probabilistic time series forecasting was proposed in [24]. It combines state-space models with deep learning. By parameterizing a per-time-series linear state-space model with a jointly learned RNN, the method retains the desired properties of state-space models such as data efficiency and interpretability while making use of the ability to learn complex patterns from raw data offered by deep learning approaches.

To improve forecasting performance, RNN is also mixed with other methods such as ETS. Such a model won the M4 forecasting competition in 2018 [25]. This competition utilized 100 000 real-life time series and incorporates all major forecasting methods, including those based on AI and ML, as well as traditional statistical ones [26]. The winning model, developed by Smyl [27], is a hybrid approach utilizing both statistical and ML features. It combined ETS with advanced LSTM, which is supported by such mechanisms as dilation, residual connections, and attention [28]–[30]. It produced the most accurate forecasts as well as the most precise prediction intervals. According to sMAPE, it was close to 10% more accurate than the combination benchmark of the competition, which is a huge improvement. For monthly data (48 000 time series), it outperformed all other 60 submissions achieving the highest accuracy according to each of the three performance measures.

The motivation for this work is as follows. Accurate load forecasts are of utmost importance to ensure a safe and efficient power system operation, increased revenues from the electricity market, and financial risk reduction. Forecast accuracy translates directly into financial performance for the energy market players and the financial impact can be measured in millions of dollars for every point of forecasting accuracy gained. MTLF is a relevant and challenging problem requiring the forecasting model to be highly flexible and deal with the stochastic data expressing nonstationarity and seasonality. In this work, we propose a state-of-the-art forecasting model for MTLF that meets these high requirements. It is based on the winning submission to the M4 competition for monthly data. It combines ETS, LSTM, and ensembling. ETS enables the model to capture the main components of the individual time series, such as seasonality and level, whereas LSTM allows nonlinear trends and cross-learning. A common learning procedure for LSTM and ETS leads to simultaneous optimization of data representation and forecasting performance. Ensembling at three levels reduces the model variance and increases generalization. Moreover, a penalized asymmetric pinball loss function can reduce the forecast bias. All these mechanisms enable the forecasting of "difficult" time series and ensure high forecasting accuracy, which was verified in the M4 competition.

The main contribution of this study includes the following two points.

1) This work empirically demonstrates that the proposed generic hybrid model using specific mechanisms of time series processing and prediction outperforms in MTLF well-established statistical and ML approaches and is on a pair with state-of-the-art domain-adjusted models combining ML and statistical approaches.
2) Time series used for MTLF share some patterns that cannot be exploited when using traditional per-series forecasting models. At the same time, the amount of data is rather small (a few dozens of monthly series). We show that despite that, a hybrid model that combines some per-series parameters with a global LSTM-style NN can perform well in this task.

The rest of the work is organized as follows. Section II describes the proposed forecasting model: its architecture, features, components, and implementation details as well as data flow and processing. Section III describes the experimental framework used to evaluate the performance of the proposed model. Finally, Section IV concludes the work.

II. Forecasting Model

The proposed model is based on the winning submission to the M4 forecasting competition 2018 for monthly data and point forecasts [27]. It is a hybrid and hierarchical forecasting model that enables ETS and advanced LSTM to be mixed into a common framework. The model architecture, its specific features, and components are described in the following.

A. Framework and Features

The proposed forecasting model is shown in Fig. 1. It is composed of the following.

1) ETS—which is a Holt–Winters-type multiplicative seasonal model. It is used for extracting two components

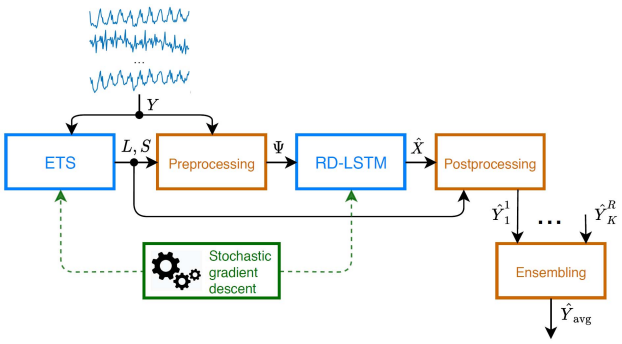

Fig. 1. Block diagram of the ETS+RD-LSTM forecasting system.

from the time series: level and seasonality. ETS loads a set of time series (Y), calculates the level and seasonal components individually for each series, and returns sets of levels (L) and seasonal components (S).

2) Preprocessing—the level and seasonal components are used for deseasonalization and adaptive normalization of the time series. The inputs to the preprocessing module are: set of time series Y and sets of level and seasonal components, L and S, respectively. The preprocessed data are divided into input and output training data and returned in training set Ψ.
3) RD-LSTM—which is residual dilated LSTM composed of four layers. Due to its recurrent nature, this model is capable of learning long-term dependencies in sequential data. RD-LSTM learns in cross-learning mode on training set Ψ. The forecasts for all time series produced by RD-LSTM are returned in set $\hat{X}$.
4) Postprocessing—the forecasts of the deseasonalized and normalized time series are "reseasonalized" and renormalized. The inputs to the postprocessing module are: forecasts $\hat{X}$ and level and seasonality sets, L and S. The output is set $\hat{Y}$, containing the forecasts for each time series.
5) Ensembling—the forecasts produced by individual models are averaged. This enhances the robustness of the method further, mitigating model and parameter uncertainty. The ensembling module receives the sets of forecasts produced by individual models, $\hat{Y}_k^r$ (k and r relate to ensemble members, see Section II-E for details), aggregates them, and returns a set of forecasts for all time series, $\hat{Y}_{\text{avg}}$.
6) Stochastic gradient descent (SGD)—the parameters of both ETS and RD-LSTM are updated by the same overall optimization procedure, SGD, with the overarching goal of minimizing forecasting errors.

The proposed model has a hierarchical structure, i.e., the data are exploited in a hierarchical manner. Both local and global time series features are extracted. The global features are learned by RD-LSTM across many time series. The specific features of each individual time series are extracted by ETS. Thus, each series has a partially unique and partially shared model.

Note the hybrid structure of the model, where statistical modeling is combined concurrently with ML algorithms. The model combines ETS, advanced LSTM, and ensembling. ETS is focused on each individual series and enables the model to capture its main components, such as seasonality and level. These components are used for time series preprocessing, normalization, and deseasonalization.

An advanced LSTM-based RNN allows nonlinear trends and cross-learning. This is an extended, multilayer version of LSTM with residual dilated LSTM blocks. The dilated recurrent skip connections and spatial shortcut path from lower layers, applied in this solution, allow the model to better capture long-term seasonal relationships and ensure more efficient training. The RD-LSTM model is trained on many time series (cross-learning). To train deep NNs, which have many parameters, cross-learning is necessary. Moreover, it enables the method to capture the shared features and components of the time series.

ETS and RD-LSTM are optimized simultaneously, i.e., the ETS parameters and the RD-LSTM weights are optimized by SGD at the same time. The same overall learning procedure optimizes the model, including data preprocessing. Therefore, the learning process includes representation learning—searching for the most suitable representations of input and output data (individually for each time series), which ensures the most accurate forecasts. It is worth noting the dynamical character of the training set that is related to representation learning. The training set is updated in each epoch of RD-LSTM learning. This is because SGD updates the ETS parameters in each epoch, and therefore, the level and seasonal components, used for preprocessing, are updated as well.

Ensembling is seen as a much more powerful regularization technique than more popular alternatives, e.g., dropout or L2-norm penalty [22]. In our case, ensembling combines individual forecasts at three levels: stage of training level, data subset level, and model level. This reduces the variance related to the stochastic nature of SGD and also related to data and parameter uncertainty.

B. Exponential Smoothing

The complex nature of a time series, e.g., nonstationarity, nonlinear trend, and seasonal variations, makes forecasting difficult and puts high demands on the models. A typical approach, in this case, is to simplify the forecasting problem by deseasonalization, detrending, or decomposition. A time series is usually decomposed into seasonal, trend, and stochastic components. The components expressing less complexity than the original time series can be modeled independently using simpler models. The most popular methods of decomposition are [31]: additive decomposition, multiplicative decomposition, X11, SEAT, and STL. Although very useful, this approach has a drawback. It separates the preprocessing from the forecasting, which results in the final solution not being optimal. Some classic statistical models, such as ETS, employ a better way; the forecasting model has a built-in mechanism to deal with seasonality. The final model uses the optimal decomposition of the time series.

It is worth mentioning that in the state-of-the-art deep learning models, the time series preprocessing can be incorporated

into the learning process as in [27] or [44]. In the latter work, there is an additional deep learning layer that learns how the data should be normalized according to their distribution instead of using fixed normalization schemes. The adaptive normalization parameters are global, i.e., the same for each time point. In [27] (this approach is also used in this study), these parameters are local, adjusted for each time point of each time series. This makes the model very flexible in processing time series of different nature, with nonlinear trend and seasonality.

In our approach, we use ETS as the preprocessing tool. ETS extracts two components from the time series: level (smoothed value) and seasonality. Then, we use these components to normalize and deseasonalize the original time series. Preprocessed time series are forecast by RD-LSTM. ETS and RD-LSTM are optimized simultaneously using SGD. Therefore, the resulting forecasting model, including data preprocessing, is optimized as a whole. This distinctive feature of the proposed approach needs to be emphasized.

The ETS model used in this study was inspired by the Holt–Winters multiplicative seasonal model. However, it has been simplified by the removal of the linear trend component. This is because the trend forecasting is the task of RD-LSTM, which is able to produce a nonlinear trend that is more valuable in our case. The updating formulas for the ETS model with a seasonal cycle length of 12 (useful for monthly data) are as follows [27]:

$$\begin{aligned} l_t &= \alpha \frac{y_t}{s_t} + (1-\alpha) l_{t-1} \\ s_{t+12} &= \beta \frac{y_t}{l_t} + (1-\beta) s_t \end{aligned} \tag{1}$$

where y_t is the time series value at time point t, l_t, and s_t are the level and seasonal components, respectively, and α, $\beta \in [0, 1]$ are smoothing coefficients.

The level equation shows a weighted average between the seasonally adjusted observation and the level for time $t-1$. The seasonal equation expresses a seasonal component for time $t+12$ as a weighted average between a new estimate of the seasonality component (y_t/l_t) and the past estimate (s_t). Fig. 2 shows an example of the monthly electricity demand time series and its level and seasonal components obtained from (1).

The ETS model parameters, 12 initial seasonal components and two smoothing coefficients for each time series, were adjusted together with RD-LSTM weights by SGD. Knowing these parameters allows the level and seasonal components to be calculated, which are then used for preprocessing: deseasonalization and normalization.

C. Preprocessing and Postprocessing

The level and seasonal components are calculated for all points of each series, which are then used for deseasonalization and adaptive normalization during the on-the-fly preprocessing. This is the most crucial element of the forecasting procedure as it determines its performance. The time series is preprocessed in each training epoch using the updated

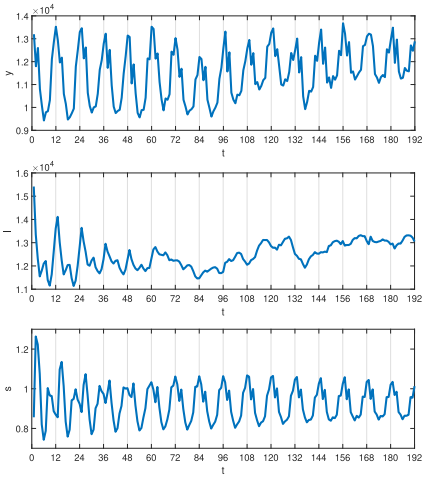

Fig. 2. Original time series and its level and seasonal components.

values of level and seasonal components. These updated values are calculated from (1), where the ETS parameters are increasingly fine-tuned in each epoch by SGD.

The time series is preprocessed using rolling windows: input and output ones. Both windows have a length of 12, which is equal to the length of both the seasonal cycle and the forecast horizon. The input window, Δ_{in}, contains 12 consecutive elements of the time series, which after preprocessing will be the RD-LSTM inputs (i.e., components of input vector $\mathbf{x}_t^{\text{in}}$). The corresponding output window, Δ_{out}, contains the next 12 consecutive elements, which after preprocessing will be the RD-LSTM outputs (i.e., components of output vector $\mathbf{x}_t^{\text{out}}$). The time series fragments inside both windows are normalized by dividing them by the last value of the level in the input window l_t^* and, then, divided further by the relevant seasonal component. As a result of this operation, we obtain positive input and output values close to one. Finally, to limit the destructive impact of outliers on the forecasts, a squashing function, log(.), is applied. The resulting preprocessing can be expressed as follows:

$$x_t = \log\left(\frac{y_t}{l_t^* s_t}\right) \tag{2}$$

where x_t is the preprocessed tth element of the time series, l_t^* is the last value of the level in input window Δ_{in}, and s_t is the tth seasonal component.

Note that normalization is adaptive and local, and the "normalizer" follows the series values. This allows us to include the current features of the series (l_t^* and s_t) in the input and output variables.

The preprocessed elements of the time series contained in the successive input and output windows can be represented by vectors as follows.

1) First pair of input and output windows

$$\mathbf{x}_1^{\text{in}} = [x_1\ x_2 \cdots x_{12}], \quad \mathbf{x}_1^{\text{out}} = [x_{13} x_{14} \cdots x_{24}].$$

2) Second pair of input and output windows

$$\mathbf{x}_2^{\text{in}} = [x_2\ x_3 \cdots x_{13}], \quad \mathbf{x}_2^{\text{out}} = [x_{14} x_{15} \cdots x_{25}],$$

3) $\cdots$

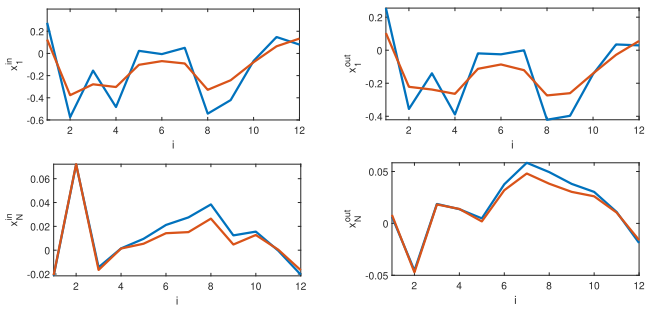

Fig. 3. Examples of the input (left) and output (right) vectors created for the time series shown in Fig. 2. Top: x-vectors for the first two cycles. Bottom: x-vectors for the last two cycles. The x-vectors created in the first training epoch in blue, in the last epoch in red.

4) Nth pair of input and output windows:

$$\mathbf{x}_N^{\text{in}} = [x_N x_{N+1} \ldots x_{N+11}]$$
$$\mathbf{x}_N^{\text{out}} = [x_{N+12} x_{N+13} \ldots x_{N+23}].$$

These vectors are included in the training subset for the ith time series: $\Phi_i = \{(\mathbf{x}_t^{\text{in}}, \mathbf{x}_t^{\text{out}}) : t = 1, 2, \ldots, N\}$. The training subsets for all M time series are combined and form the training set $\Psi = \{\Phi_1, \Phi_2, \ldots, \Phi_M\}$, which is used for RD-LSTM cross-learning. Note the dynamic character of the training set. It is updated in each epoch because the level and seasonal components in (2) are updated.

Fig. 3 shows the input and output vectors for the time series, which is shown in Fig. 2. The top shows the corresponding input and output x-vectors representing the first two seasonal cycles of the time series. The bottom shows the input and output x-vectors representing the last two seasonal cycles. It can be seen from this figure that the x-vectors express patterns of the time series fragments after filtering out both level and seasonality. This pattern representation of the time series has been used successfully in earlier studies concerning ML forecasting models, especially similarity-based models [32]. Different definitions of the time series patterns can be found in [32]. However, these definitions are fixed, while in this work, we use dynamic patterns that change during learning (compare the patterns in the first and last training epochs in Fig. 3).

RD-LSTM operates on preprocessed time series values, x_t. In the postprocessing step, the forecasts generated by RD-LSTM, $\hat{x}_t$, need to be "unwound" in the following way:

$$\hat{y}_t = \exp(\hat{x}_t) l_t^* s_t. \tag{3}$$

Note that both level l_t^* and seasonal component s_t in (3), which are necessary to calculate $\hat{y}_t$ from $\hat{x}_t$, are known. They are determined from (1) on the basis of the time series history.

D. Residual Dilated LSTM

LSTM is a special kind of RNN capable of learning long-term dependencies in sequential data [33]. A common LSTM block is composed of a memory cell that can maintain its state over time, and three nonlinear "regulators," called gates, which control the flow of information inside the block. A typical LSTM block is shown in Fig. 4. In this diagram, $\mathbf{h}_t$ and $\mathbf{c}_t$ denote the hidden state and the cell state at time step t, respectively. The cell state contains information learned from the previous time steps. Information can be added to or removed from the cell state using the gates: input gate (i), forget gate (f), and output gate (o). At each time step t, the block uses the past state of the network, i.e., $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, and the input $\mathbf{x}_t$ to compute output $\mathbf{h}_t$ and updated cell state $\mathbf{c}_t$. The hidden and cell states are recurrently connected back to the block input. All of the gates are controlled by the hidden state of the past cycle and the input x-vector. Most modern studies incorporate many of the improvements that have been made to the LSTM architecture since its original formulation [34].

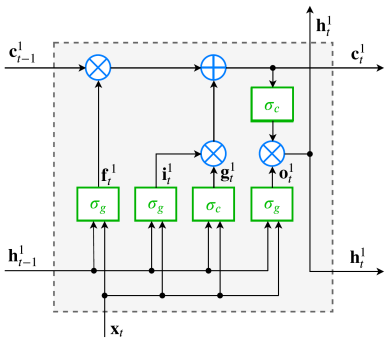

Fig. 4. LSTM block.

Detailed mathematical expressions describing LSTM block are given in the following. The cell state at time step t is

$$\mathbf{c}_t^1 = \mathbf{f}_t^1 \otimes \mathbf{c}_{t-1}^1 + \mathbf{i}_t^1 \otimes \mathbf{g}_t^1 \tag{4}$$

where operator $\otimes$ denotes the Hadamard product (elementwise product) and superscript 1 refers to the first layer of RD-LSTM network, where we use the standard LSTM block.

The hidden state at time step t is given by

$$\mathbf{h}_t^1 = \mathbf{o}_t^1 \otimes \sigma_c(\mathbf{c}_t^1) \tag{5}$$

where the state activation function σ_c is a hyperbolic tangent function.

Formulas related to the gates are as follows:

$$\mathbf{f}_t^1 = \sigma_g(\mathbf{W}_f^1 \mathbf{x}_t + \mathbf{V}_f^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_f^1) \tag{6}$$
$$\mathbf{i}_t^1 = \sigma_g(\mathbf{W}_i^1 \mathbf{x}_t + \mathbf{V}_i^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_i^1) \tag{7}$$
$$\mathbf{g}_t^1 = \sigma_c(\mathbf{W}_g^1 \mathbf{x}_t + \mathbf{V}_g^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_g^1) \tag{8}$$
$$\mathbf{o}_t^1 = \sigma_g(\mathbf{W}_o^1 \mathbf{x}_t + \mathbf{V}_o^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_o^1) \tag{9}$$

where $\mathbf{W}$, $\mathbf{V}$, and $\mathbf{b}$ are input weights, recurrent weights, and biases, respectively, and σ_g is a gate sigmoid activation function $(1 + e^{-x})^{-1}$.

In our study, we also use a dilated residual version of LSTM (RD-LSTM). Dilated RNN architecture was proposed in [28] as a solution to tackle three major challenges of RNN when learning on long sequences: complex dependencies, vanishing and exploding gradients, and efficient parallelization. It is characterized by multiresolution dilated recurrent skip connections. Moreover, it reduces the number of parameters needed and enhances training efficiency significantly in tasks involving very long-term dependencies. A dilated LSTM block

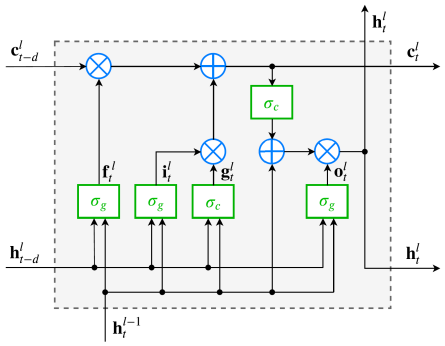

Fig. 5. RD-LSTM block.

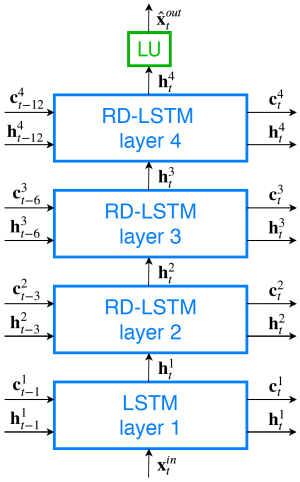

Fig. 6. RD-LSTM network architecture.

receives as input states, not the last ones, $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, but earlier states, $\mathbf{c}_{t-d}$ and $\mathbf{h}_{t-d}$, where $d > 1$ is a dilation. Thus, to compute the current states of the LSTM block, the last $d-1$ states are skipped. Usually, multiple dilated recurrent layers are stacked with hierarchical dilations to construct a system, which learns the temporal dependencies of different scales at different layers. In [28], it was shown that this solution can reliably improve the ability of recurrent models to learn long-term dependence in problems from different domains. It seems that dilated LSTM can be particularly useful for seasonal time series, where the relationships between the series elements have a cyclical character. This character can be incorporated into the model by dilations related to seasonality.

A residual version of LSTM was proposed in [29]. A standard memory cell, which learns long-term dependencies of sequential data, provides a temporal shortcut path to avoid vanishing or exploding gradients in the temporal domain. The residual LSTM provides an additional spatial shortcut path from lower layers for efficient training of deep LSTM architectures. To avoid a conflict between spatial- and temporal-domain gradient flows, residual LSTM separates the spatial shortcut path from the temporal one. This gives greater flexibility to deal with vanishing or exploding gradients.

Fig. 5 shows a residual dilated LSTM block that was used in this study. We denote this block by RD-LSTM. In this figure, $\mathbf{h}_t^{l-1}$ is a shortcut path from the $(l-1)$th layer that is added to updated cell state $\mathbf{c}_t$ processed by function σ_c. Our implementation of the residual LSTM is a simplified version of the original one. The peephole connections are removed as well as linear transformations of the shortcut path $\mathbf{h}_t^{l-1}$ and transformed cell state $\sigma_c(\mathbf{c}_t^l)$. These transformations are not necessary because, in our case, the dimensions of $\mathbf{h}_t^{l-1}$ and $\sigma_c(\mathbf{c}_t^l)$ match that of $\mathbf{h}_t^l$.

The mathematical expressions describing the RD-LSTM block are as follows:

$$\mathbf{c}_t^l = \mathbf{f}_t^l \otimes \mathbf{c}_{t-d}^l + \mathbf{i}_t^l \otimes \mathbf{g}_t^l \tag{10}$$
$$\mathbf{h}_t^l = \mathbf{o}_t^l \otimes \left(\sigma_c\left(\mathbf{c}_t^l\right) + \mathbf{h}_t^{l-1}\right) \tag{11}$$
$$\mathbf{f}_t^l = \sigma_g\left(\mathbf{W}_f^l \mathbf{h}_t^{l-1} + \mathbf{V}_f^l \mathbf{h}_{t-d}^l + \mathbf{b}_f^l\right) \tag{12}$$
$$\mathbf{i}_t^l = \sigma_g\left(\mathbf{W}_i^l \mathbf{h}_t^{l-1} + \mathbf{V}_i^l \mathbf{h}_{t-d}^l + \mathbf{b}_i^l\right) \tag{13}$$
$$\mathbf{g}_t^l = \sigma_c\left(\mathbf{W}_g^l \mathbf{h}_t^{l-1} + \mathbf{V}_g^l \mathbf{h}_{t-d}^l + \mathbf{b}_g^l\right) \tag{14}$$
$$\mathbf{o}_t^l = \sigma_g\left(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{V}_o^l \mathbf{h}_{t-d}^l + \mathbf{b}_o^l\right) \tag{15}$$

where superscript l indicates the layer number (from 2 to 4 in our case, see next) and d is a dilation (3, 6, or 12 in our case, see next).

The proposed RD-LSTM architecture, which is a result of painstaking experimentation on M4 monthly data (48 000 time series), is shown in Fig. 6. It is composed of four recurrent layers and a linear unit LU. The first layer consists of the standard LSTM block shown in Fig. 4. The subsequent three layers consist of RD-LSTM blocks (see Fig. 5) with increasing dilations $d = 3$, 6, and 12. The last element is a linear unit that transforms the output of the last layer, $\mathbf{h}_t^4$, into the forecast of the output x-vector

$$\hat{\mathbf{x}}_t^{\text{out}} = \mathbf{W}_x \mathbf{h}_t^4 + \mathbf{b}_x. \tag{16}$$

The learnable parameters of RD-LSTM are: input weights $\mathbf{W}$, recurrent weights $\mathbf{V}$, and biases $\mathbf{b}$. They are tuned in the cross-learning mode simultaneously with the ETS parameters using SGD. The length of the cell and hidden states, m, the same for all layers, was selected on the training set (see Section III for details).

RD-LSTM processes the temporal information as follows. The model sequentially learns on the consecutive pairs of input and output vectors, $\mathbf{x}_t^{\text{in}}$ and $\mathbf{x}_t^{\text{out}}$, respectively, for $t = 1, 2, \ldots$. For each $\mathbf{x}_t^{\text{in}}$, RD-LSTM produces the forecast of $\mathbf{x}_t^{\text{out}}$ taking into account previous states, $\mathbf{h}_{t-1}^1$, $\mathbf{c}_{t-1}^1$, $\mathbf{h}_{t-3}^2$, $\mathbf{c}_{t-3}^2$, $\mathbf{h}_{t-6}^3$, $\mathbf{c}_{t-6}^3$, $\mathbf{h}_{t-12}^4$, and $\mathbf{c}_{t-12}^4$ (see Fig. 6). These states contain past information about time series provided not only from the last $t-1$ cycle but also from cycles $t-3$, $t-6$, and $t-12$. This mechanism can be useful for the seasonal time series to introduce additional information about long-term seasonal relationships. In each tth learning cycle, RD-LSTM modifies its parameters depending on the forecast error for the tth training pair and generates new states, $\mathbf{h}_t^l$ and $\mathbf{c}_t^l$, for the next cycles (in fact, due to residual connections, the new hidden states are also used in the current cycle). Thus, the temporal information flows from the previous cycles to the next ones using past hidden and cell states. We can control this information changing the dilations.

Note that an input is not a scalar but a vector representing a sequence of the time series of length 12, i.e., a seasonal period. It allows the RD-LSTM to be exposed to the immediate history of the series directly. An output is a vector representing the whole forecast sequence of length 12.

When the output x-pattern is determined from (16), the forecast monthly demands are calculated from (3). In the latter equation, $\hat{x}_t$ is the tth component of vector $\hat{\mathbf{x}}_{\text{avg}}^{\text{out}}$.

E. Ensembling

Ensembling is a well-known approach to increase the predictive performance of statistical and ML models. Ensemble methods combine in some fashion multiple learning algorithms to produce a common response, hopefully improving accuracy and stability compared to a single learner. The key issue in ensemble learning is ensuring diversity of learners [43]. A right tradeoff between performance and diversity of learners decides about ensemble learning success. In our case, for ensembling, we can use several sources of diversity in RD-LSTM. The first one is the stochastic training process using SGD. The second one is similar to bagging, i.e., training each learner using a randomly drawn subset of the training set. The third one is training ensemble members using different initial values of the parameters. Thus, the forecasts generated by the RD-LSTM model are ensembled at three levels as follows.

1) *Stage of Training Level:* Averaging forecasts produced by L most recent training epochs.
2) *Data Subset Level:* Averaging forecasts from a pool of K forecasting models, which learns on the subsets of the training set.
3) *Model Level:* Averaging forecasts from R independent runs of the pool of K models produced on the data subset level.

The idea behind using the averaging forecasts produced in a few most recent training epochs (first level of ensembling) is that averaging has the effect of calming down the noisy SGD optimization process. SGD uses mini-batches of training samples to estimate the actual gradients. The resulting searching process operates on the approximated gradients that make the trajectory noisy. Averaging the forecasts obtained in the most recent epochs, when the algorithm converges around the local minimum, may reduce the effect of stochastic searching and produce more accurate forecasts.

At the data subset level, the forecasts from K models that learn on the subsets of the training set, $\Psi_1, \Psi_2, \ldots, \Psi_K$, are averaged. The training set, $\Psi = \{\Phi_1, \Phi_2, \ldots, \Phi_M\}$, is composed of subsets Φ_i containing the training samples for the ith time series. To create the training Ψ-subsets, first, a set of M time series is split randomly into K subsets of similar size: $\Theta_1, \Theta_2, \ldots, \Theta_K$. The kth Ψ-subset contains the Φ-subsets for all time series excluding those in Θ_k, i.e., $\Psi_k = \Psi \backslash \{\Phi_i\}_{i \in \Theta_k}$. Each of K models learns on its own training subset, Ψ_k, and generates forecasts for the time series included in Ψ_k. Then, the $K-1$ forecasts for each time series produced by the pool of K models are averaged.

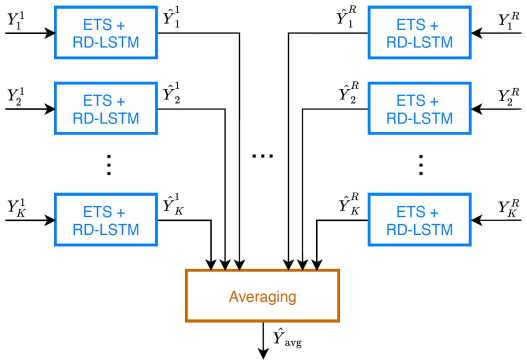

Fig. 7. Ensembling.

The last level of ensembling simply averages the forecasts for each time series generated in R independent runs of a pool of K models. In each run, the training subsets Ψ_k are created anew.

Note that the diversity of learners, which is a key property that governs the ensemble performance [35], has various sources in our proposed approach. They include: 1) data uncertainty: learning on mini-batches and learning on different Ψ-subsets of the training set and 2) parameter uncertainty: learning using different initial values of the model parameters in each run.

The last two ensembling levels are shown in Fig. 7. In this figure, K training Ψ-subsets are created. The set of time series included in the kth Ψ-subsets in the rth run is denoted by Y_k^r, and the set of forecasts generated by the model in this case is denoted by $\hat{Y}_k^r$. For each time series, $R(K-1)$ forecasts are averaged. This is shown as a joint operation for levels 2 and 3 in the figure and can be expressed as

$$\hat{\mathbf{y}}_{\text{avg}} = \frac{1}{R(K-1)} \sum_{r=1}^{R} \sum_{k=1}^{K-1} \hat{\mathbf{y}}_{r,k} \tag{17}$$

where K is the size of the pool of models, R is the number of runs, and $\hat{\mathbf{y}}_{r,k}$ is the forecast y-vector generated as the kth one in the rth run.

In this work, we use a simple average for ensembling, but other functions, such as median, mode, or trimmed mean, could be applied [35]. As shown in [36], a simple average of forecasts often outperforms more complicated weighting schemes.

Note that the $K \cdot R$ models created at the data subset levels in R runs can be trained simultaneously. Thus, the proposed forecasting system is suitable to be implemented in parallel.

F. Loss Function

As a loss function, we used pinball loss operating on normalized and deseasonalized RD-LSTM outputs and actuals

$$L_t = \begin{cases} (x_t - \hat{x}_t)\tau, & \text{if } x_t \geq \hat{x}_t \\ (\hat{x}_t - x_t)(1-\tau), & \text{if } \hat{x}_t > x_t \end{cases} \tag{18}$$

where $\tau \in (0, 1)$ controls the function asymmetry.

When $\tau = 0.5$, the loss function is symmetrical and penalizes positive and negative deviations equally. When the model

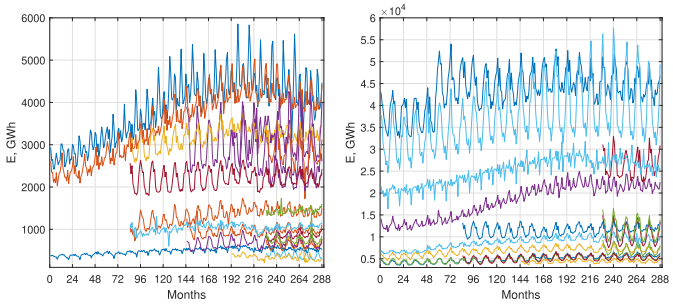

Fig. 8. Monthly electricity demand time series for 35 European countries.

tends to have a positive or negative bias, we can reduce the bias by introducing τ smaller or larger than 0.5, respectively. Thus, the asymmetric pinball function, penalizing positive and negative deviations differently, allows the method to deal with bias. It is worth noting that pinball loss is commonly employed in quantile regression and probabilistic forecasting [37].

The experience gained during the M4 competition shows that the smoothness of the level time series influences the forecasting accuracy substantially. It turned out that when the input to the RD-LSTM was smooth, the RD-LSTM focused on predicting the trend, instead of overfitting on some spurious, seasonality-related patterns. A smooth-level curve also means that the seasonal components absorbed the seasonality properly. To deal with the wiggliness of a level curve, a penalized loss function was introduced as follows.

1) Calculate the logarithms of the quotients of the neighboring points of the level time series: $d_t = \log(l_{t+1}/l_t)$.
2) Calculate differences of the above: $e_t = d_{t+1} - d_t$.
3) Square and average them for each series.

The resulting penalized loss function related to a given time series takes the form

$$L = \frac{1}{T}\sum_{t=1}^{T} L_t + \lambda \frac{2}{T-2}\sum_{t=1}^{T-2} \log\left(\frac{l_{t+2}l_t}{l_{t+1}^2}\right) \tag{19}$$

where T is the number of forecasts and λ is a regularization parameter that determines how much to penalizes the wiggliness of a level curve.

The level wiggliness penalty affected the performance of the method significantly and contributed greatly to winning the M4 competition [27].

III. Experimental Study

This section presents the results of applying the proposed forecasting model to the monthly electricity demand forecasting for 35 European countries. The data were obtained from the ENTSO-E repository (www.entsoe.eu). The time series have different lengths: 24 years (11 countries), 17 years (6 countries), 12 years (4 countries), 8 years (2 countries), and 5 years (12 countries). The last year of data is 2014. The time series are presented in Fig. 8. As can be seen from this figure, monthly electricity demand time series exhibit different levels, nonlinear trends, strong annual cycles, and variable variances. The shapes of yearly cycles change over time.

The forecasting model as well as comparative models were applied to forecast 12 monthly demands for the last year of data, 2014. The data from previous years were used for hyperparameter selection and learning. Specifically, for the selection of hyperparameters, the models were trained on the time series fragments up to 2012 and validated in 2013 (during the M4 competition, a very strong correlation between errors for the test period and the last period of training data was observed). Then, the selected optimal hyperparameters were used to construct the model for 2014. The selected hyperparameters were as follows:

1) number of epochs: 16;
2) learning rate: 10^{-3};
3) length of the cell and hidden states: $m = 40$;
4) asymmetry parameter in pinball loss: $\tau = 0.4$;
5) regularization parameter: $\lambda = 50$;
6) ensembling parameters: $L = 5$, $K = 3$, and $R = 3$.

Due to the stochastic nature of ETS+RD-LSTM, the results reported in this article take averages over 30 independent trials (learning sessions). The model was implemented in C++ relying on the DyNet library [38]. It was compiled in Visual Studio 2017 (Windows 10) and run in parallel on an eight-core CPU (AMD Ryzen 7 1700, 3.0 GHz, 32 GB RAM).

A. Comparative Models

The proposed model was compared with state-of-the-art models based on ML as well as classical statistical models, such as ARIMA and ETS. All ML models except LSTM use a pattern representation of time series [14]. Patterns that express preprocessed repetitive sequences in a time series ensure input and output data unification through trend filtering and variance equalization. Consequently, the relationship between input and output data is simplified and the transformed forecasting problem can be solved using simple models.

The comparative models are described in detail in [14] and outlined in the following.

1) *k-Nearest Neighbor Weighted Regression Model (k-NNw):* It estimates a vector-valued regression function as an average of output patterns in a varying neighborhood of a query pattern. A weighting function allows the model to consider the similarity between the query pattern and its nearest neighbors. The model hyperparameters are: input pattern length and number of nearest neighbors k. Linear weighting function was used.
2) *Fuzzy Neighborhood Model (FNM):* In this case, the regression model is similar to k-NNw except that it aggregates not only k-nearest neighbors of the query pattern but also all training patterns. The weighting function takes the form of a membership function (Gaussian-type function), which assigns to each training pattern a degree of membership to the query pattern neighborhood. The model hyperparameters are: input pattern length and membership function width.
3) *Nadaraya—Watson estimator (N-WE):* It is a kernel regression model belonging to the same category of nonparametric models as k-NNw and FNM. N-WE estimates the regression function as a locally weighted average, using a kernel function (Gaussian) as a weighting

function. The model hyperparameters are: input pattern length and kernel bandwidth parameters.

4) *General Regression NN (GRNN) model:* This is a four-layer NN with Gaussian nodes centered at training patterns. The node outputs express similarities between the query pattern and the training patterns. These outputs are treated as the weights of the training patterns in the regression model. The model hyperparameters are: input pattern length and bandwidth parameter for nodes.
5) *Multilayer perceptron (MLP)* [39] with a single hidden layer and sigmoidal neurons. It used for learning the Levenberg–Marquardt method with Bayesian regularization to prevent overfitting. The MLP hyperparameters are: input pattern length and number of hidden nodes. We use MATLAB R2018a implementation of MLP (function `feedforwardnet` from Neural Network Toolbox).
6) *Adaptive Neuro-Fuzzy Inference System (ANFIS) [40]:* The initial membership function parameters in the premise parts of rules are determined using fuzzy c-means clustering. A hybrid learning method is applied for ANFIS training, which uses a combination of the least squares for consequent parameters and backpropagation gradient descent method for premise parameters. The ANFIS hyperparameters are: input pattern length and number of rules. The MATLAB R2018a implementation of ANFIS was used (function `anfis` from Fuzzy Logic Toolbox).
7) *k-NNw+ETS, FNM+ETS, N-WE+ETS, GRNN+ETS, MLP+ETS*, and *ANFIS+ETS*—variants of the above models with output patterns encoded with variables describing the current features of the time series. These features are the mean value of the series in the seasonal cycle and its dispersion. To postprocess the forecast output pattern, the coding variables are predicted for the next period using ETS. We use R implementation of ETS (see next).
8) *LSTM*, where the responses are the training sequences with values shifted by one time step (a sequence-to-sequence regression LSTM network). For multiple time steps, after one step was predicted, the LSTM state was updated. Previous prediction was used as input to LSTM, producing a forecast for the next time step. LSTM was optimized using Adam (adaptive moment estimation) optimizer. The length of the hidden state was the only hyperparameter to be tuned. Other hyperparameters remain at their default values. The experiments were carried out using MATLAB R2018a implementation of LSTM (function `trainNetwork` from Neural Network Toolbox).
9) *ARIMA*—ARIMA$(p, d, q)(P, D, Q)_{12}$ model implemented in function `auto.arima` in R environment (package `forecast`). This function implements automatic ARIMA modeling, which combines unit root tests, minimization of the Akaike information criterion (AICc), and maximum likelihood estimation to obtain the optimal ARIMA model [31].
10) *ETS*—exponential smoothing state space model [42] implemented in function `ets` (R package `forecast`). This implementation includes many types of ETS models depending on how the seasonal, trend, and error components are considered. They can be expressed additively or multiplicatively, and the trend can be damped or not. As in the case of `auto.arima`, `ets` returns the optimal model estimating its parameters using AICc.

TABLE I
RESULTS COMPARISON AMONG THE PROPOSED AND COMPARATIVE MODELS

Model	Median *APE*	*MAPE*	*IQR*	*RMSE*
k-NNw	2.89	4.99	3.85	368.79
FNM	2.88	4.88	4.26	354.33
N-WE	2.84	5.00	3.97	352.01
GRNN	2.87	5.01	4.02	350.61
k-NNw+ETS	2.71	4.47	3.52	327.94
FNM+ETS	**2.64**	4.40	3.46	321.98
N-WE+ETS	2.68	**4.37**	3.36	**320.51**
GRNN+ETS	**2.64**	4.38	3.51	324.91
MLP	2.97	5.27	3.84	378.81
MLP+ETS	3.11	4.80	4.12	358.07
ANFIS	3.56	6.18	4.87	488.75
ANFIS+ETS	3.54	6.32	4.26	464.29
LSTM	3.73	6.11	4.50	431.83
ARIMA	3.32	5.65	5.24	463.07
ETS	3.50	5.05	4.80	374.52
ETS+RD-LSTM	2.86	4.46	**3.33**	351.43

The models, k-NNw, FNM, N-WE, and GRNN are also known as pattern similarity-based forecasting models (PSFMs) because the forecast is constructed by aggregating the training output patterns using similarity between the query pattern and training input patterns [14]. All the model hyperparameters mentioned above were selected on the training set in grid search procedures. Due to the stochastic nature of the NN-based models, i.e., MLP, ANFIS, and LSTM, 100 independent learning sessions for these models were performed and the final errors were calculated as averages over 100 trials.

B. Results

Table I shows the results of forecasting for the proposed and comparative models: median of absolute percentage error (APE), mean APE (MAPE), interquartile range of APE as a measure of the forecast dispersion, and root-mean-square error (RMSE). These are values averaged over 35 countries. The lowest errors are for "+ETS" PSBMs, where the coding variables are predicted using ETS. The errors for ETS+RD-LSTM are slightly higher but lower than for all other models. To confirm that the differences in errors are statistically significant, we performed the Wilcoxon rank-sum test. Results indicated that there is no difference at the 5% significance level in MAPE between ETS+RD-LSTM, all PSFMs, MLP, and MLP+ETS.

More detailed results are shown in Figs. 9 and 10. Fig. 9 shows MAPE for each country. As can be seen from this figure, ETS+RD-LSTM is one of the most accurate models in most cases. Fig. 10 shows MAPE for each month of the

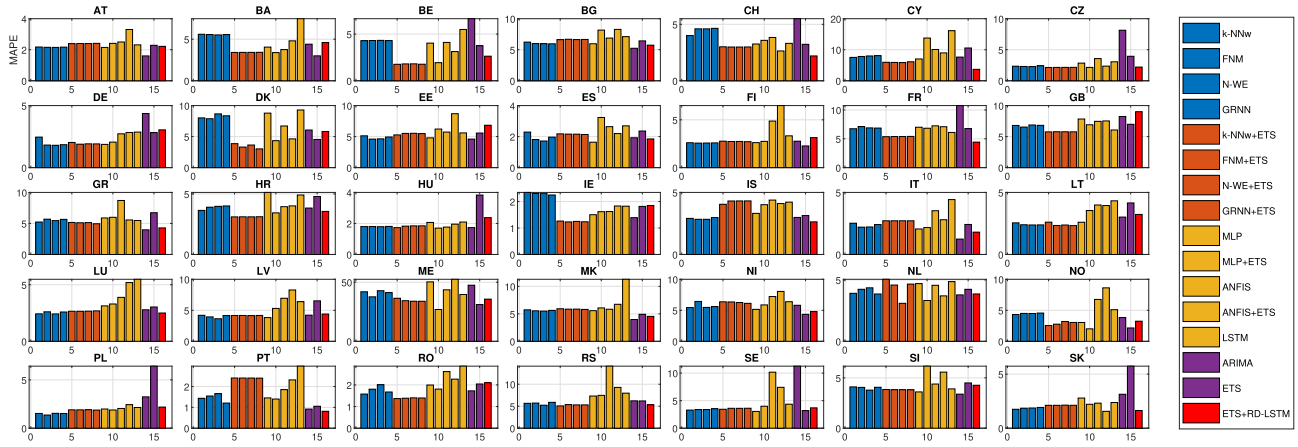

Fig. 9. MAPE for each country.

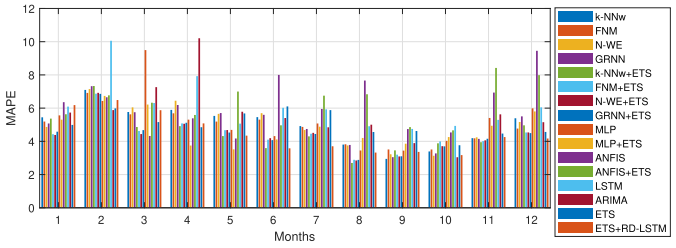

Fig. 10. MAPE for each month of the forecast period.

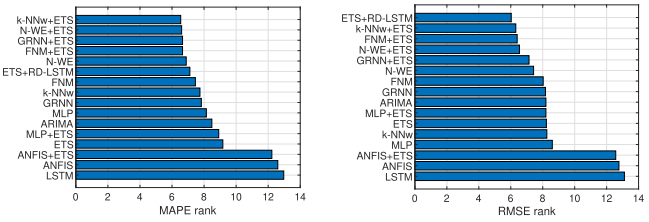

Fig. 11. Rankings of the models.

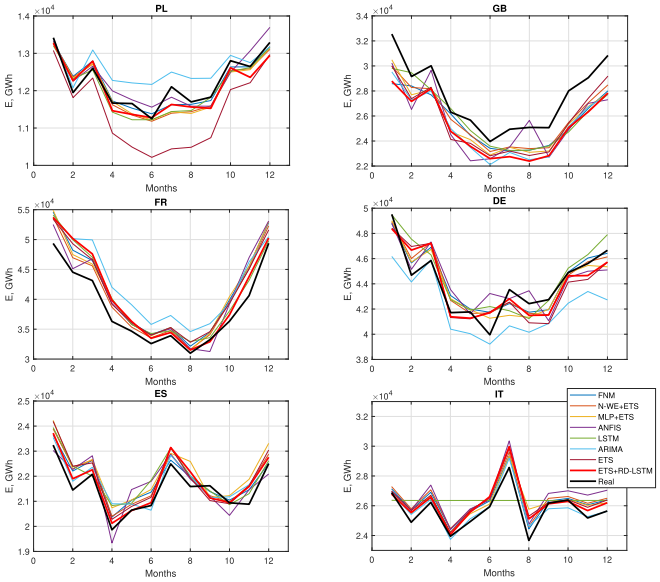

Fig. 12. Real and forecast monthly electricity demand for selected countries.

forecast period. Note lower errors for months 8–10 and higher for months 1–4 and 12. ETS+RD-LSTM achieved better results than most of the comparative models for months 2, 5–8, 10, and 12. For months 1 and 3, it achieved higher errors than most competitors.

Rankings of the models are shown in Fig. 11. These are based on average ranks of the models in the rankings for individual countries. The left of Fig. 11 shows the ranking based on MAPE and the right of Fig. 11 shows the ranking based on RMSE. Note the high position of ETS+RD-LSTM. It is in the first position in RMSE ranking and the sixth position in MAPE ranking. Note that in the latter case, the difference between the first six positions is very small.

Examples of forecasts produced by the selected models for six countries are shown in Fig. 12. For PL data, most models, including ETS+RD-LSTM, do not exceed MAPE $=$ 2%, which should be considered a very good result. Similar results were achieved for ES, IT, and DE. In these cases, MAPE for ETS+RD-LSTM was 1.61% (most accurate model), 2.12% (third most accurate model), and 2.29%. For GB, the forecasts are underestimated. This results from the fact that demand went up unexpectedly in 2014 despite the downward trend observed in the previous period from 2010 to 2013. The reverse situation for FR caused a slight overestimation of forecasts. For GB data, ETS+RD-LSTM with MAPE $=$ 8.52% was the least accurate model, and for FR data, with MAPE $=$ 5.49%, it was one the most accurate models.

Table II shows the basic descriptive statistics of the percentage errors (PEs) and Fig. 13 shows the empirical probability density functions of PEs for the selected models. The PE distributions are similar to normal, but tests for the assessment of normality (Jarque–Bera test and Lilliefors test) do not confirm this. In all cases, the forecasts are overestimated, having a negative PE mean. Note that ETS+RD-LSTM is the least biased model. Negative values of skewness indicate the left-skewed PE distributions, and high kurtosis values indicate leptokurtic distributions where the probability mass is concentrated around the mean.

C. Ablation Study

We performed an ablation study to find out what is the impact of the model components on its performance. We used

TABLE II
DESCRIPTIVE STATISTICS OF PERCENTAGE ERRORS

	Mean	Median	Std	Skewness	Kurtosis
k-NNw	-1.87	-1.08	10.43	-5.14	44.56
FNM	-2.03	-1.22	9.34	-4.16	35.14
N-WE	-1.91	-1.18	10.82	-5.41	48.94
GRNN	-1.87	-1.16	10.60	-5.34	48.11
k-NNw+ETS	-1.25	-0.20	9.00	-4.40	37.30
FNM+ETS	-1.26	**-0.11**	8.80	-4.75	41.71
N-WE+ETS	-1.26	-0.17	8.68	-4.63	40.75
GRNN+ETS	-1.26	**-0.11**	8.61	-4.42	38.38
MLP	-1.37	-0.68	11.88	-7.52	109.64
MLP+ETS	-1.71	-1.03	**7.32**	-1.55	**11.83**
ANFIS	-2.51	-1.43	11.37	-4.35	34.93
ANFIS+ETS	-1.30	-0.40	12.65	**-0.96**	39.37
LSTM	-3.12	-1.81	9.49	-2.86	22.21
ARIMA	-2.35	-1.03	13.62	-9.01	119.20
ETS	-1.04	-0.31	7.97	-1.89	13.52
ETS+RD-LSTM	**-0.74**	-0.25	9.18	-5.27	48.42

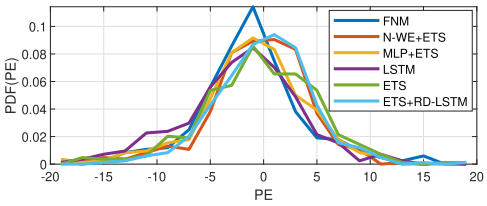

Fig. 13. PDF of the PEs.

TABLE III
MAPE FOR DIFFERENT RD-LSTM ARCHITECTURES AND DILATIONS

#layers	4	4	3	3	3	2	2	2	1
Dilations	1,3,6,12	1,3,6,9	1,6,12	1,4,8	1,3,6	1,12	1,6	1,3	1
MAPE	4.46	**4.43**	4.46	**4.45**	4.48	4.52	4.49	4.49	4.49

MAPE metric as performance criterion and Wilcoxon rank-sum test with the 5% significance level for comparing the base model with its lean versions. Results presented in the following are calculated as averages over 30 trials.

1) Ensembling: We start our study with the base model excluding individual levels of ensembling: stage of training level (–L1), data subset level (–L2), model level (–L3), and finally excluding all ensembling levels (–L1L2L3). The results (MAPE) were as follows: –L1: 4.46%, –L2: 4.41%, –L3: 4.47%, and –L1L2L3: 4.48%. The Wilcoxon test indicated that there is no difference in MAPE between the base model and its variants without ensembling.

2) Residual Connections: After removing residual connections, the RD-LSTM architecture is composed of the four stacked standard LSTM blocks (without shortcut paths from lower layers). The error without residual connections increased to 4.74% and is statistically higher than error for the base model.

3) Dilations: The architecture of the base model is composed of the four LSTM layers with dilations $d = 1$, 3, 6, and 12. Table III shows MAPE for other architectures. As can be seen from this table, MAPE is on a similar level for all configurations. Two architectures (bolded) yield statistically lower errors.

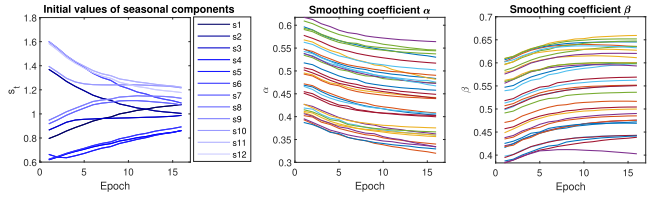

Fig. 14. ETS parameters during training.

4) Loss Function: The asymmetric pinball loss function (18), penalizing positive and negative deviations differently, allows the method to deal with bias. When we use the symmetric pinball loss, with $\tau = 0.5$, the positive and negative deviations are penalized equally. In such a case, the error increased to 4.51%. However, this increase is not statistically significant as well as the error increase when we use MSE instead of the pinball loss (MAPE = 4.59%). When we removed a penalty term from the penalized loss function (19), the error admittedly decreased to 4.39%, but it was statistically indistinguishable from the base model error.

5) ETS: ETS is a built-in component of the proposed model providing a mechanism for time series preprocessing. As mentioned in Section II-B, this mechanism adaptively adjusts the ETS parameters, i.e., initial values of the 12 seasonal components and two smoothing coefficients. SGD updates them in each training epoch for each time series individually. This is shown in Fig. 14. Fig. 14 (left) shows the initial values of the seasonal components for PL time series, and Fig. 14 (middle and right) shows the smoothing coefficients for all countries. As can be seen from this figure, initial seasonal components shrink toward 1. It was characteristic not only of PL data but also of all other countries. Note that coefficients α decrease and coefficients β increase with training epochs. When we remove the ETS component from the model, the time series are not adaptively preprocessed. In such a case, the RD-LSTM module faces a more difficult task. This results in an increase in error to 4.71%.

D. Discussion

The results presented in Section III-B show that our proposed ETS+RD-LSTM model is very effective at solving the MTLF problem. It has the high expressive power to solve nonlinear stochastic forecasting problems. In our study, we used the architecture that won the M4 competition without major modifications. The simulation study showed that it can compete with other models, including classical statistical methods, ML, and hybrid approaches.

The ablation study revealed that some components of the base winning model are crucial and others can be excluded or simplified when we apply ETS+RD-LSTM to MTLF. Namely, it turned out that the following conditions hold.

1) Ensembling did not meet the expectations set before it. Excluding all ensembling levels, we achieved similar results to the base model with three levels of ensembling. Note that without ensembling, the learning procedure simplifies: instead of training $K \cdot R$ ETS+RD-LSTM models, we train only one model. Ensembling contributed significantly to the success in the M4 competition. However, it should be noted that

in our case, the model learned on only 35 time series, whereas in M4, it learned from a data set containing 48 000 monthly time series. For such large data sets, ensembling showed its advantages.

2) Residual connections play an important role in the model operation. Besides speeding up the training process, they introduce additional information from the lower layers to the upper ones. This allowed the model to increase accuracy in MTLF.
3) The number of layers and dilation hyperparameters is dependent strongly on the forecasting problem. We used the wining M4 architecture, but simpler architectures turned out to be just as accurate. The most parsimonious architecture is the one with only one layer.
4) The asymmetric pinball loss function allows the model to reduce the forecast bias. In our case, we achieved slightly lower errors when applying the asymmetric function than the symmetric one. The pinball loss function can be replaced easily by other loss functions, e.g., MSE. The penalization term in the loss function smooths the level component and thus reduces overfitting, which leads to improved performance. However, in MTLF, it did not matter. The results with and without penalization were at a similar level.
5) ETS is a crucial built-in component of the model that adaptively preprocesses time series and enables the model to learn their representations. RD-LSTM without time series preprocessing losses its performance.

It must be remembered that ETS+RD-LSTM was designed as a universal model for forecasting time series with annual seasonality from diverse domains. The conducted experiments have shown that this universal architecture achieves state-of-the-art performance also in MTLF. However, the base architecture can be simplified without losing performance as the ablation study has shown. The reason for this is that in our case, we use a small amount of data to train ETS+RD-LSTM (only 35 short time series). In such a case, the model cannot fully use its powerful mechanisms that reveal themselves when it is trained on big data. These mechanisms include cross-learning transfering and shearing individual learning, ensembling combining individual forecasts, asymmetric penalized loss function that prevents overfitting and reduces bias, and dilation in multilayered architecture that enables the model to capture long-term seasonal relationships.

Disadvantages of the model include a complex architecture and a large number of parameters and hyperparameters to tune. In the base variant, the architecture consists of ETS component and stacked LSTM components. The complex architecture with hundreds of parameters to adjust needs the time-consuming training. As in the case of other deep NNs, the learning process using gradient-based algorithms suffers from the sensitivity to initial values of training parameters, convergence to local minima, and the vanishing/exploding gradient problem.

IV. Conclusion

In this work, we proposed and empirically validated a new architecture for MTLF. It was inspired by the winning submission to the M4 forecasting competition 2018, which combines ETS, advanced LSTM, and ensembling. The model has a hierarchical structure composed of a global part learned across many time series (weights of the LSTM) with a time series specific part (ETS smoothing coefficients and initial seasonal components). Using SGD, it learns a mapping from input to output vectors and time series representations at the same time. ETS-inspired formulas extract the main components of individual time series to be used for deseasonalization and normalization. Preprocessed time series are forecast using residual dilated LSTM. Due to the introduction of dilated recurrent skip connections and a spatial shortcut path from lower layers, LSTM is able to capture better long-term seasonal relationships and ensure more efficient training. To deal with forecast bias, we used a pinball loss function with a parameter controlling its asymmetry. The model is equipped with two regularization mechanisms: penalized loss function and three-level ensembling.

We applied the proposed model to the monthly electricity demand forecasting for 35 European countries. The results demonstrated its state-of-the-art performance and competitiveness with classical models such as ARIMA and ETS as well as state-of-the-art models based on ML.

References

[1] F. Apadula, A. Bassini, A. Elli, and S. Scapin, "Relationships between meteorological variables and monthly electricity demand," *Appl. Energy*, vol. 98, pp. 346–356, Oct. 2012.
[2] E. Dogan, "Are shocks to electricity consumption transitory or permanent? Sub-national evidence from turkey," *Utilities Policy*, vol. 41, pp. 77–84, Aug. 2016.
[3] L. Suganthi and A. A. Samuel, "Energy models for demand forecasting—A review," *Renew Sust Energy Rev*, vol. 16, no. 2, pp. 1223–1240, 2002.
[4] E. H. Barakat, "Modeling of nonstationary time-series data. Part II. Dynamic periodic trends," *Int. J. Electr. Power Energy Syst.*, vol. 23, no. 1, pp. 63–68, Jan. 2001.
[5] E. González-Romera, M. A. Jaramillo-Morán, and D. Carmona-Fernández, "Monthly electric energy demand forecasting with neural networks and Fourier series," *Energy Convers. Manage.*, vol. 49, no. 11, pp. 3135–3142, Nov. 2008.
[6] J.-F. Chen, S.-K. Lo, and Q. Do, "Forecasting monthly electricity demands: An application of neural networks trained by heuristic algorithms," *Information*, vol. 8, no. 1, p. 31, Mar. 2017.
[7] M. Gavrilas, "Medium-term load forecasting with artificial neural network models," in *Proc. 16th Int. Conf. Exhib. Electr. Distrib. (CIRED)*, Jun. 2001, p. 5.
[8] E. Doveh, P. Feigin, D. Greig, and L. Hyams, "Experience with FNN models for medium term power demand predictions," *IEEE Trans. Power Syst.*, vol. 14, no. 2, pp. 538–546, May 1999.
[9] P. Pełka and G. Dudek, "Medium-term electric energy demand forecasting using generalized regression neural network," *Proc. Conf. Inf. Syst. Archit. Technol. ISAT*, in Advances in Intelligent Systems and Computing, vol. 853. Cham, Switzerland: Springer, 2019, pp. 218–227.
[10] T. Ahmad and H. Chen, "Potential of three variant machine-learning models for forecasting district level medium-term and long-term energy demand in smart grid environment," *Energy*, vol. 160, pp. 1008–1020, Oct. 2018.
[11] P.-C. Chang, C.-Y. Fan, and J.-J. Lin, "Monthly electricity demand forecasting based on a weighted evolving fuzzy neural network approach," *Int. J. Electr. Power Energy Syst.*, vol. 33, no. 1, pp. 17–27, Jan. 2011.
[12] S.-M. Baek, "Mid-term load pattern forecasting with recurrent artificial neural network," *IEEE Access*, vol. 7, pp. 172830–172838, 2019.
[13] W. Zhao, F. Wang, and D. Niu, "The application of support vector machine in load forecasting," *J. Comput.*, vol. 7, no. 7, pp. 1615–1622, Jul. 2012.

[14] G. Dudek and P. Pełka, "Pattern similarity-based machine learning methods for mid-term load forecasting: A comparative study," 2020, *arXiv:2003.01475*. [Online]. Available: http://arxiv.org/abs/2003.01475

[15] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," 2019, *arXiv:1909.00590*. [Online]. Available: http://arxiv.org/abs/1909.00590

[16] K. Yan, X. Wang, Y. Du, N. Jin, H. Huang, and H. Zhou, "Multi-step short-term power consumption forecasting with a hybrid deep learning strategy," *Energies*, vol. 11, no. 11, p. 3089, Nov. 2018.

[17] J. Bedi and D. Toshniwal, "Empirical mode decomposition based deep learning for electricity demand forecasting," *IEEE Access*, vol. 6, pp. 49144–49156, 2018.

[18] H. Zheng, J. Yuan, and L. Chen, "Short-term load forecasting using EMD-LSTM neural networks with a xgboost algorithm for feature importance evaluation," *Energies*, vol. 10, no. 8, p. 1168, Aug. 2017.

[19] A. Narayan and K. W. Hipel, "Long short term memory networks for short-term electric load forecasting," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 2573–2578.

[20] J.-F. Toubeau, J. Bottieau, F. Vallee, and Z. De Greve, "Deep learning-based multivariate probabilistic forecasting for short-term scheduling in power markets," *IEEE Trans. Power Syst.*, vol. 34, no. 2, pp. 1203–1215, Mar. 2019.

[21] T. Zia and S. Razzaq, "Residual recurrent highway networks for learning deep sequence prediction models," *J. Grid Comput.*, vol. 18, no. 1, pp. 169–176, Jun. 2018.

[22] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *Proc. ICLR*, 2020. [Online]. Available: https://openreview.net/forum?id=r1ecqn4YwB

[23] D. Salinas, V. Flunkert, and J. Gasthaus, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," 2017, *arXiv:1704.04110*. [Online]. Available: http://arxiv.org/abs/1704.04110

[24] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Proc. NeurIPS*, vol. 31, 2018, pp. 7785–7794.

[25] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: Results, findings, conclusion and way forward," *Int. J. Forecasting*, vol. 34, no. 4, pp. 802–808, Oct. 2018.

[26] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: 100,000 time series and 61 forecasting methods," *Int. J. Forecasting*, vol. 36, no. 1, pp. 54–74, Jan. 2020.

[27] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *Int. J. Forecasting*, vol. 36, no. 1, pp. 75–85, Jan. 2020.

[28] S. Chang *et al.*, "Dilated recurrent neural networks," 2017, *arXiv:1710.02224*. [Online]. Available: http://arxiv.org/abs/1710.02224

[29] J. Kim, M. El-Khamy, and J. Lee, "Residual LSTM: Design of a deep recurrent architecture for distant speech recognition," 2017, *arXiv:1701.03360*. [Online]. Available: http://arxiv.org/abs/1701.03360

[30] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 2627–2633.

[31] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Melbourne, VIC, Australia, Mar. 2020.

[32] G. Dudek, "Pattern similarity-based methods for short-term load forecasting—Part 1: Principles," *Appl. Soft Comput.*, vol. 37, pp. 277–287, Dec. 2015.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[35] F. Petropoulos, R. J. Hyndman, and C. Bergmeir, "Exploring the sources of uncertainty: Why does bagging for time series forecasting work?" *Eur. J. Oper. Res.*, vol. 268, no. 2, pp. 545–554, Jul. 2018.

[36] F. Chan and L. L. Pauwels, "Some theoretical results on forecast combinations," *Int. J. Forecasting*, vol. 34, no. 1, pp. 64–74, Jan. 2018.

[37] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile estimation," *J. Mach. Learn. Res.*, vol. 7, no. Jul, pp. 1231–1264, 2006.

[38] G. Neubig *et al.*, "DyNet: The dynamic neural network toolkit," 2017, *arXiv:1701.03980*. [Online]. Available: http://arxiv.org/abs/1701.03980

[39] P. Pełka and G. Dudek, "Pattern-based forecasting monthly electricity demand using multilayer perceptron," in *Proc. Conf. Artif. Intell. Soft Comput. ICAISC* in Lecture Notes in Artificial Intelligence, vol. 11508. Cham, Switzerland: Springer, 2019, pp. 663–672.

[40] P. Pełka and G. Dudek, "Neuro-fuzzy system for medium-term electric energy demand forecasting," in *Proc. Conf. Inf. Syst. Archit. Technol. ISAT* in Advances in Intelligent Systems and Computing, vol. 655. Cham, Switzerland: Springer, 2018, pp. 38–47.

[41] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0194889.

[42] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting With Exponential Smoothing: The State Space Approach*. Berlin, Germany: Springer-Verlag, 2008.

[43] G. Brown, J. L. Wyatt, and P. Tiňo, "Managing diversity in regression ensembles," *J. Mach. Learn. Res.*, vol. 6, pp. 1621–1650, Dec. 2005.

[44] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Deep adaptive input normalization for time series forecasting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3760–3765, Sep. 2020.

Grzegorz Dudek received the Ph.D. degree in electrical engineering from the Częstochowa University of Technology, Częstochowa, Poland, in 2003, and the Habilitation degree in computer science from the Lodz University of Technology, Łódź, Poland, in 2013.

He is currently an Associate Professor with the Department of Electrical Engineering, Częstochowa University of Technology. He is the author of two books concerning machine learning methods for load forecasting and evolutionary algorithms for unit commitment and over 100 scientific articles. His research interests include pattern recognition, machine learning, artificial intelligence, and their application to practical classification, regression, forecasting, and optimization problems.

Dr. Dudek won third in the Global Energy Forecasting Competition 2014 (price forecasting track).

Paweł Pełka received the M.Sc. degree in informatics from the Częstochowa University of Technology, Częstochowa, Poland, in 2016. He is currently pursuing the Ph.D. degree with the Faculty of Mechanical Engineering and Computer Science, with a focus on pattern-based midterm load forecasting issues.

He is currently an Assistant Professor with the Faculty of Electrical Engineering, Częstochowa University of Technology. He is an author of several scientific articles. His research interests include data analysis, machine learning, pattern recognition, artificial intelligence, and their usage in time series forecasting problems.

Slawek Smyl received the M.Sc. degree in physics from Jagiellonian University, Kraków, Poland, in 1988, and the M.Eng. degree in information technology from RMIT University, Melbourne, VIC, Australia, in 1997.

He is currently a Staff Data Scientist with Uber Technologies, working in the area of time series forecasting.

Mr. Smyl has ranked highly in forecasting competitions, i.e., winning the Computational Intelligence in Forecasting International Time Series Competition 2016; got a third place in the Global Energy Forecasting Competition in 2017; and won the M4 Forecasting Competition in 2018.