Recurrent Neural Networks for Forecasting Time Series with Multiple Seasonality: A Comparative Study

Grzegorz Dudek[1][0000−0002−2285−0327], Slawek Smyl[2][0000−0003−2548−6695], and Paweł Pełka[1][0000−0002−2609−811X]

[1] Electrical Engineering Faculty, Czestochowa University of Technology, Poland
{grzegorz.dudek,pawel.pelka}@pcz.pl
[2] Meta, 1 Hacker Way, Menlo Park, CA 94025, USA
slawek.smyl@gmail.com

Abstract. This paper compares recurrent neural networks (RNNs) with different types of gated cells for forecasting time series with multiple seasonality. The cells we compare include classical long short term memory (LSTM), gated recurrent unit (GRU), modified LSTM with dilation, and two new cells we proposed recently, which are equipped with dilation and attention mechanisms. To model the temporal dependencies of different scales, our RNN architecture has multiple dilated recurrent layers stacked with hierarchical dilations. The proposed RNN produces both point forecasts and predictive intervals (PIs) for them. An empirical study concerning short-term electrical load forecasting for 35 European countries confirmed that the new gated cells with dilation and attention performed best.

Keywords: LSTM · Multiple seasonality · RNN · Short-term load forecasting · Time series forecasting.

1 Introduction

Forecasting time series (TS) with multiple seasonality is a challenging problem. To solve it, a forecasting model has to deal with short- and long-term dynamics as well as a trend and variable variance. Classical statistical methods such as autoregressive moving average (ARMA) and exponential smoothing methods can be extended to multiple seasonal cycles [1, 2] but they suffer from many drawbacks. The most important of these are: their linear nature, limited adaptability, limited ability to model complex seasonal patterns, problems with capturing long-term dependencies and problems with introducing exogenous variables.

To improve the ability of statistical models to capture multiple seasonality, various approaches have been applied, such as extending the model with Fourier terms [3, 4], TS decomposition [5] and local modeling [6, 7]. Machine learning (ML) gives additional opportunities to the models and makes them more flexible. The main idea behind ML is to learn from past observations any inherent

structures, patterns or anomalies within the data, with the objective of generating future values for the series. The most popular ML models in the field of forecasting are neural networks (NNs) [8] as they can flexibly model complex nonlinear relationships with minimum a-priori assumptions and reflect process variability in uncertain dynamic environments. They offer learning of representation, cross-learning on massive datasets and modeling temporary relationships in sequential data. In particular, RNNs, which were designed for sequential data such as TS and text data, are extremely useful for forecasting. They form a directed graph along a temporal sequence which is able to exhibit temporal dynamic behavior using their internal state (memory) to process sequences of inputs.

Modern RNNs, such as LSTM and GRU, are capable of learning both short and long-term dependencies in TS [9]. They are equipped with recurrent cells that can maintain their states over time and, using nonlinear "regulators" called gates, can control the flow of information inside the cell. Recent works have reported that gated RNNs provide high accuracy in forecasting and outperform most of the statistical and ML methods, such as ARIMA, support vector machine, and shallow NNs [10]. A comparison of RNNs on multiple seasonality forecasting problems performed in [11] showed that LSTM, GRU and classical Elman RNN demonstrate comparable performance but are relatively slow in terms of training time due to the time-consuming backpropagation through the time procedure. To improve the learning capability and forecasting performance facing RNN, different mechanisms have been used such as residual connections [12] and dilated architecture [13], which solves the major challenges of RNN when learning on long sequences: i.e. complex dependencies, vanishing and exploding gradients, and efficient parallelization. Hybrid solutions have also been proposed combining RNN with TS decomposition [14] or other methods such as exponential smoothing. One such hybrid model won the reputed M4 forecasting competition in 2018, showing impressive performance [15].

Motivated by the superior performance of RNN in TS forecasting, in this study, we compare RNNs with different recurrent cells. We consider a problem of univariate forecasting TS with multiple seasonality on the example of short-term electrical load forecasting (STLF). We propose a stacked hierarchical RNN architecture trained globally across all series and equipped with recurrent cells of different types. We normalize TS input data and encode output data using coding variables determined from recent history. This is to better capture the current dynamics of the process. Such prepossessing has proven successful in other forecasting models for multiple seasonality, see [6, 16, 17].

The contribution of this study is as follows:

1. We propose a new RNN architecture for forecasting TS with multiple seasonality. It is composed of three dilated recurrent layers stacked with hierarchical dilations to deal with multiple seasonality. It uses a combined asymmetrical loss function which enables the model to produce both point forecasts and PIs and also to reduce the forecast bias.

2. We compare five types of gated recurrent cells: classical LSTM and GRU, modified LSTM with dilation, and two new cells we proposed recently, which are equipped with dilation and attention mechanisms.
3. We empirically demonstrate on real data for the electricity demand for 35 European countries that our proposed model copes successfully with complex seasonality. The new attentive dilated recurrent cell significantly outperforms its competitors in terms of accuracy.

The remainder of the paper is organised as follows. Section 2 describes the forecasting problem and data representation. Section 3 presents the recurrent cells and Section 4 describes RNN architecture. Section 5 describes the results of experiments and discusses our findings. Finally, Section 6 concludes the paper.

2 Forecasting Problem and Data Representation

In this study, as an example of forecasting time series with multiple seasonality, we consider a problem of STLF. The hourly load time series, $\{z_\tau\}_{\tau=1}^{M}$, express triple seasonality: yearly, weekly and daily (see [18] for details, where such time series are analysed). Our goal is to forecast the daily profile (24 hours) for the next day based on historical loads (univariate problem).

As input information, we introduce a weekly profile, which precedes the forecasted day. This profile is represented by the input pattern defined as follows:

$$\mathbf{x}_t = \frac{\mathbf{z}_t^w - \overline{z}_t^w}{\mathrm{std}(z_t^w)} \tag{1}$$

where $\mathbf{x}_t \in \mathbb{R}^{168}$ is the t-th weekly pattern, $\mathbf{z}_t^w \in \mathbb{R}^{168}$ is the original sequence of the t-th week, and $\overline{z}_t^w$ and $\mathrm{std}(z_t^w)$ are its mean and standard deviation, respectively.

Note that (1) expresses standardization of the weekly sequence. Thus the weekly sequences for $t = 1, ..., N$ are unified, i.e. they are centered around zero with a unit variance. This operation filters out the trend and yearly seasonality.

An output pattern represents a forecasted daily sequence as follows:

$$\mathbf{y}_t = \frac{\mathbf{z}_t^d - \overline{z}_t^w}{\mathrm{std}(z_t^w)} \tag{2}$$

where $\mathbf{y}_t \in \mathbb{R}^{24}$ is the t-th daily pattern and $\mathbf{z}_t^d \in \mathbb{R}^{24}$ is the forecasted sequence (following directly weekly sequence $\mathbf{z}_t^w$ which is encoded in $\mathbf{x}_t$).

Note that in (2), we encode the daily sequence using the mean and standard deviation of the preceding week. This enables us to decode the forecasted pattern, $\hat{\mathbf{y}}$, into the real sequence as follows:

$$\hat{\mathbf{z}}^d = \hat{\mathbf{y}}\mathrm{std}(z^w) + \overline{z}^w \tag{3}$$

where $\overline{z}^w$ and $\mathrm{std}(z^w)$ are coding variables determined on the basis of the historical weekly sequence represented by query pattern $\mathbf{x}$.

Following [18], to introduce more input information related to the forecasted sequence, we extend the input vector with the following components: $\log_{10}(\bar{z}_t^w)$, which informs about the level of the time series, $\mathbf{d}_t^w \in \{0,1\}^7, \mathbf{d}_t^m \in \{0,1\}^{31}$ and $\mathbf{d}_t^y \in \{0,1\}^{52}$, which are binary one-hot vectors encoding day of the week, day of the month and week of the year for the forecasted day. The extended input pattern takes the form:

$$\mathbf{x}'_t = [\mathbf{x}_t, \log_{10}(\bar{z}_t), \mathbf{d}_t^w, \mathbf{d}_t^m, \mathbf{d}_t^y] \tag{4}$$

The paired input and output patterns constitute the training set, $\{(\mathbf{x}'_i, \mathbf{y}_i)\}_{i=1}^N$. The proposed model is trained in cross-learning mode, i.e. on many time series [15], which enables it to capture the shared features of the individual series and prevents over-fitting. The training sets for all L time series are combined: $\Phi = \Phi_1 \cup ... \cup \Phi_L$.

3 Recurrent Cells

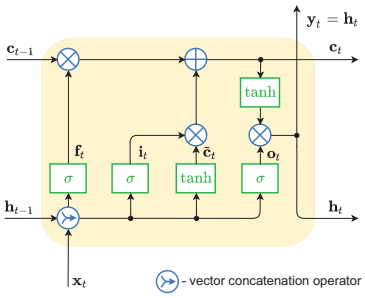
In our study, we explore RNNs with different gated recurrent cells. They include classical cells such as LSTM and GRU, modified LSTM, i.e. dilated LSTM (dLSTM), and two new solutions proposed recently, dRNNCell and adRNNCell.

3.1 LSTM

LSTM was proposed in [19] for learning problems related to sequential data. The main idea behind LSTM is a memory cell that carries relevant information throughout the processing of the sequence, and nonlinear gating units that regulate the information flow in the cell. Due to the memory, long-term temporal relationships can be captured and the effects of short-term memory can be reduced, i.e. even information from the earlier time steps can make its way to later time steps. Moreover, in LSTM, unlike in simple RNNs, the optimization problem with vanishing gradients was reduced, which improved learning capabilities.

Fig. 1 shows a diagram of LSTM. LSTM uses two states: a cell state, $\mathbf{c}_t$, and a hidden state, $\mathbf{h}_t$. The states contain information learned from the previous time steps. At each time step t, information is added to or removed from the cell state. These updates are controlled using three gates, which in fact are layers of learned nonlinear transformations. They comprise: input gate (i), forget gate (f) and output gate (o). All of the gates receive the hidden state of the past cycle and the current time series sequence as inputs. They can learn what information is relevant to keep or forget during training. At time step t, the cell uses the recent states, $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, and the input sequence, $\mathbf{x}_t$, to compute new updated states $\mathbf{c}_t$ and $\mathbf{h}_t$. The hidden and cell states are recurrently connected back to the cell input. The new hidden state, $\mathbf{h}_t$, has two functions. It controls the gating mechanism in the next step and it is treated as the cell output, $\mathbf{y}_t$, which goes to the next layer.

The compact form of the equations describing LSTM are shown in Fig. 1, where: $\mathbf{W}$ and $\mathbf{V}$ are learned weight matrices, $\mathbf{b}$ are learned bias vectors, $\otimes$ denotes the Hadamard product and σ is a logistic sigmoid function.

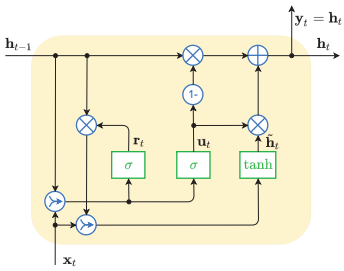


$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1} + \mathbf{b}_f)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{V}_i\mathbf{h}_{t-1} + \mathbf{b}_i)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\mathbf{h}_{t-1} + \mathbf{b}_o)$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$
$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t)$$
$$\mathbf{y}_t = \mathbf{h}_t$$

Fig. 1. LSTM.

3.2 GRU

In comparison to LSTM, in GRU the cell state was eliminated so the hidden state is used to both store information and control the gating mechanism [20]. GRU only has two gates, a reset gate (r) and an update gate (u). The update gate acts in a similar way as the forget and input gates in LSTM. It decides what information to remove and what new information to add. The reset gate decides how much past information to forget. The output gate was eliminated. The gating mechanism of GRU and the corresponding equations are shown in Fig. 2.



$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{V}_r\mathbf{h}_{t-1} + \mathbf{b}_r)$$
$$\mathbf{u}_t = \sigma(\mathbf{W}_u\mathbf{x}_t + \mathbf{V}_u\mathbf{h}_{t-1} + \mathbf{b}_u)$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{V}_h(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{b}_h)$$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \otimes \mathbf{h}_{t-1} + \mathbf{u}_t \otimes \tilde{\mathbf{h}}_t$$
$$\mathbf{y}_t = \mathbf{h}_t$$

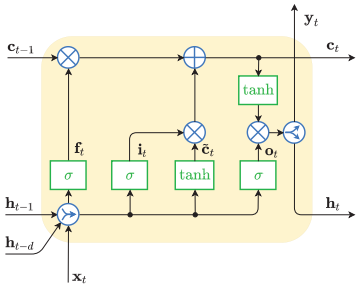
Fig. 2. GRU.

3.3 dLSTM

To improve the modeling of long-term dependencies in time series, we propose a dilated LSTM cell (Fig. 3). Our modification comes down to two elements.

First, in addition to the hidden state $\mathbf{h}_{t-1}$, we introduce a delayed hidden state, $\mathbf{h}_{t-d}$, $d > 1$. This allows the data processing in time t to be controlled using not only information from the recent state but also using direct information from the delayed state. This can be useful for seasonal time series, in which case the dilation can correspond to the period of seasonal variations. Second, the output hidden state, $\mathbf{h}'_t$, is split into "real output" $\mathbf{y}_t$, which goes to the next layer, and a controlling output $\mathbf{h}_t$, which is an input to the gating mechanism in the following time steps. This solution was inspired by [21]. The size of the c-state is equal to the summed sizes of h-state and y-output, i.e. $s_c = s_h + s_y$.

The equations corresponding to dLSTM are shown in Fig. 3, where: $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{U}$ are learned weight matrices, $\mathbf{b}$ are learned bias vectors, and s_h, s_y are the lengths of hidden state and output vectors, respectively.



$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{h}_{t-d} + \mathbf{b}_f)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{V}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{h}_{t-d} + \mathbf{b}_i)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{h}_{t-d} + \mathbf{b}_o)$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{h}_{t-d} + \mathbf{b}_c)$$
$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tilde{\mathbf{c}}_t$$
$$\mathbf{h}'_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t)$$
$$\mathbf{h}_t = [h'_{t,1}, ..., h'_{t,s_h}]$$
$$\mathbf{y}_t = [h'_{t,s_h+1}, ..., h'_{t,s_h+s_y}]$$

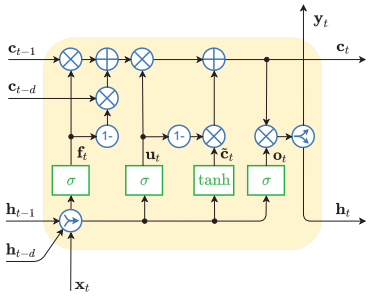
Fig. 3. dLSTM.

3.4 dRNNCell

A dilated recurrent NN cell, dRNNCell, was introduced in [18] as a combination of GRU and LSTM cells, see Fig. 4. It was designed to operate as part of a multilayer dilated RNN [13]. Its output is split into $\mathbf{y}_t$ and $\mathbf{h}_t$ as in dLSTM.

As in LSTM, dRNNCell uses two states, i.e. c-state and h-state. But, unlike LSTM, dRNNCell is fed by both most recent states, $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, and delayed states, $\mathbf{c}_{t-d}$ and $\mathbf{h}_{t-d}$, $d > 1$. dRNNCell is equipped with three gates, which transform nonlinearly input vectors using logistic sigmoid function. They comprise fusion (f), update (u), and output (o) gates. A candidate c-state, $\tilde{\mathbf{c}}_t$, is produced by transforming input vectors using tanh nonlinearity. The operation of the cell is described by the equations shown in Fig. 4.

Note that the c-state is a weighted combination of past c-states and new candidate state $\tilde{\mathbf{c}}_t$ computed in the current step. Update vector, $\mathbf{u}_t$, decides in what proportion the old and new information are mixed in the c-state, while fusion vector $\mathbf{f}_t$ decides about the contribution of recent and delayed c-states in the new state.



$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{h}_{t-d} + \mathbf{b}_f)$$
$$\mathbf{u}_t = \sigma(\mathbf{W}_u\mathbf{x}_t + \mathbf{V}_u\mathbf{h}_{t-1} + \mathbf{U}_u\mathbf{h}_{t-d} + \mathbf{b}_u)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{h}_{t-d} + \mathbf{b}_o)$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{h}_{t-d} + \mathbf{b}_c)$$
$$\mathbf{c}_t = \mathbf{u}_t \otimes (\mathbf{f}_t \otimes \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \otimes \mathbf{c}_{t-d}) + (1 - \mathbf{u}_t) \otimes \tilde{\mathbf{c}}_t$$
$$\mathbf{h}'_t = \mathbf{o}_t \otimes \mathbf{c}_t$$
$$\mathbf{h}_t = [h'_{t,1}, ..., h'_{t,s_h}]$$
$$\mathbf{y}_t = [h'_{t,s_h+1}, ..., h'_{t,s_h+s_y}]$$

Fig. 4. dRNNCell.

3.5 adRNNCell

An attentive dilated recurrent NN cell, adRNNCell, was proposed in [22] as an extended version of dRNNCell. It combines two dRNNCells to obtain a more efficient cell, which is able to preprocess dynamically the sequence data. It is equipped with an attention mechanism for weighting the input information.

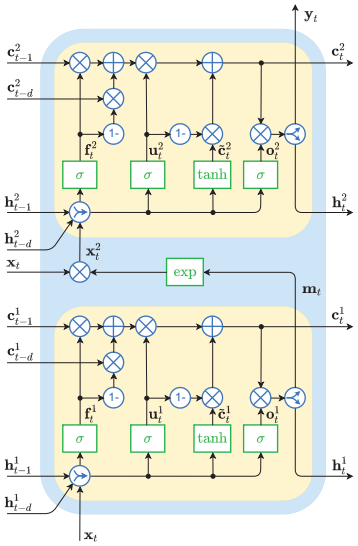
Fig. 5 shows adRNNCell composed of lower and upper dRNNCells. The former produces attention vector $\mathbf{m}_t$ of the same length as the input vector $\mathbf{x}_t$. The components of $\mathbf{m}_t$, after processing by exp function, are treated as weights for the inputs collected in $\mathbf{x}_t$. The weighted inputs, $\mathbf{x}_t^2$, feed the upper cell. The goal of such an attention mechanism is to dynamically strengthen or weaken particular inputs depending on their relevance. Note that this process is dynamic, the weights are adjusted to the current inputs at time t. Both cells, lower and upper, learn simultaneously. Based on the weighted input vector, $\mathbf{x}_t^2$, the upper cell predicts vector $\mathbf{y}_t$.

The mathematical model describing adRNNCell is shown in Fig. 5.

4 RNN Architecture

In this study, we adopt RNN architecture from [22]. It is composed of three single-layer blocks, see Fig. 6. In each block, the cells are dilated differently, i.e. 2, 4 and 7, respectively. Delayed connections enable the direct input into the cell of information from a few time steps ago. This can be useful in modeling seasonal dependencies. To model the temporal dependencies of different scales, our architecture has multiple dilated recurrent layers stacked with hierarchical dilations. It also uses ResNet-style shortcuts between blocks to improve the learning process [12].

To reduce input dimmensionality, the calendar variables, $\mathbf{d}_t^w$, $\mathbf{d}_t^m$ and $\mathbf{d}_t^y$, are embedded using a linear layer into d-dimensional continuous vector $\mathbf{d}_t$. The second linear layer at the top of stacked recurrent layers, produces the point forecasts, $\hat{\mathbf{y}}_t$, and two vectors of quantiles, a lower one, $\underline{\hat{\mathbf{y}}}_t \in \mathbb{R}^{24}$, and an upper one, $\hat{\overline{\mathbf{y}}}_t \in \mathbb{R}^{24}$. These quantiles of assumed orders, $\underline{q}$ and $\overline{q}$, define the PI.



$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{h}_{t-d} + \mathbf{b}_f)\\
\mathbf{u}_t &= \sigma(\mathbf{W}_u\mathbf{x}_t + \mathbf{V}_u\mathbf{h}_{t-1} + \mathbf{U}_u\mathbf{h}_{t-d} + \mathbf{b}_u)\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{h}_{t-d} + \mathbf{b}_o)\\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{h}_{t-d} + \mathbf{b}_c)\\
\mathbf{c}_t &= \mathbf{u}_t \otimes (\mathbf{f}_t \otimes \mathbf{c}_{t-1} + (1-\mathbf{f}_t) \otimes \mathbf{c}_{t-d})\\
&\quad + (1-\mathbf{u}_t) \otimes \tilde{\mathbf{c}}_t\\
\mathbf{x}^2_t &= \mathbf{x}_t \otimes \mathbf{m}_t\\
\mathbf{h}^{1\prime}_t &= \mathbf{o}^1_t \otimes \mathbf{c}^1_t\\
\mathbf{h}^1_t &= [h^{1\prime}_{t,1}, ..., h^{1\prime}_{t,s_h}]\\
\mathbf{m}_t &= [h^{1\prime}_{t,s_h+1}, ..., h^{1\prime}_{t,s_h+s_m}]\\
\mathbf{h}^{2\prime}_t &= \mathbf{o}^2_t \otimes \mathbf{c}^2_t\\
\mathbf{h}^2_t &= [h^{2\prime}_{t,1}, ..., h^{2\prime}_{t,s_q}]\\
\mathbf{y}_t &= [h^{2\prime}_{t,s_q+1}, ..., h^{2\prime}_{t,s_q+s_y}]
\end{aligned}
$$

Fig. 5. adRNNCell.

To enable our RNN to learn both point forecasts and PI quantiles, we employ the following loss function [18]:

$$L = \rho(y, \hat{y}_{q^*}) + \gamma(\rho(y, \hat{y}_{\underline{q}}) + \rho(y, \hat{y}_{\overline{q}})) \tag{5}$$

where ρ is a pinball loss:

$$\rho(y, \hat{y}_q) = \begin{cases} (y - \hat{y}_q)q & \text{if } y \geq \hat{y}_q \\ (y - \hat{y}_q)(q-1) & \text{if } y < \hat{y}_q \end{cases} \tag{6}$$

$q \in (0,1)$ is a quantile order, y is an actual value (standardized), $\hat{y}_q$ is a forecasted value of q-th quantile of y, $q^* = 0.5$ corresponds to the median, $\underline{q} \in (0, q^*)$ and $\overline{q} \in (q^*, 1)$ correspond to the lower and upper bound of PI, respectively, and $\gamma \geq 0$ is a parameter controlling the impact of the components related to PI on the loss function, typically between 0.1 and 0.5.

The first component in (5) is a symmetrical loss for the point forecast, while the second and third components are asymmetrical losses for the quantiles. The asymmetry level, which determines PI, results from the quantile orders. For example, we obtain a 90% symmetrical PI for $\underline{q} = 0.05$ and $\overline{q} = 0.95$.

Remarks:

1. In Section 5, we compare RNN with the different cell types which were described in Section 3. Note that RNN shown in Fig. 6 requires cells equipped with both recent and delayed connections. Classical LSTM and GRU are not equipped with delayed inputs. RNN with these cells are considered in two variants: (i) LSTM1, GRU1 – the delayed connections are removed and cells

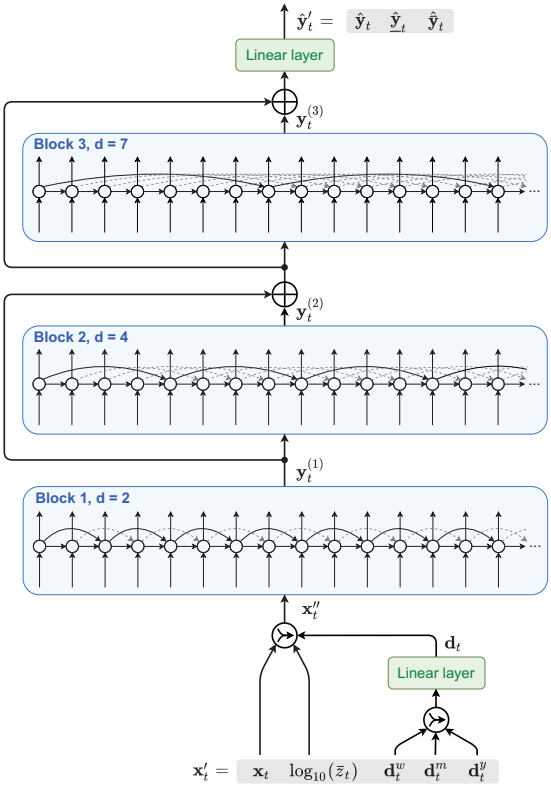


Fig. 6. RNN architecture.

are fed with only recent inputs $t-1$, and (ii) LSTM2, GRU2 – the recent connections are removed and cells are fed with only delayed inputs, $t-2$, $t-4$ or $t-7$, depending in the layer.

2. The pinball loss gives the opportunity to reduce the forecast bias by penalizing positive and negative deviations differently. When the model tends to have a positive or negative bias, we can reduce the bias by introducing q^* smaller or larger than 0.5, respectively (see [15, 23]).

5 Experimental Study

We compare the performance of the proposed RNN with different recurrent cells on STLF problems for 35 European countries. The data, collected from ENTSO-E repository (www.entsoe.eu/data/power-stats), concerns real-world hourly electrical load time series. The data period is from 2006 to 2018 but a large amount of data is missing in this period (about 60% of the countries have complete data). For 35 countries, the data provides a variety of time series with triple seasonality expressing different properties such as different levels, trends, variance and daily

shapes (see Section II in [18] where these time series are analysed). We treat data from 2018 as test data. We predict daily load profiles for each day of the test period and each country with the exception of three countries. For these three countries, due to missing data, the test periods were shorter, i.e. for Estonia and Italy (missing last month of data) and Latvia (missing last two months of data).

The RNNs were optimized on data from the period 2006-17. As performance metrics we use: mean absolute percentage error (MAPE), median of APE (MdAPE), interquartile range of APE (IqrAPE), root mean square error (RMSE), mean PE (MPE), and standard deviation of PE (StdPE). Below, we report results for an ensemble of five RNNs (average of five RNN runs). We use a similar training and optimization setup as in [22]. The key hyperparameters were: $s_c = 250$, $s_h = s_q = s_y = 125$, $q^* = 0.5$, $\underline{q} = 0.05$, $\overline{q} = 0.95$, $\gamma = 0.3$, number of epochs: 10, learning rates: $3 \cdot 10^{-3}$ (epochs 1-5), 10^{-3} (epoch 6), $3 \cdot 10^{-4}$ (epoch 7), 10^{-4} (epochs 8-10), batch size: 2 (epochs 1-3), 5 (epochs 4-10).

Table 1 displays the forecasting quality metrics averaged over the 35 countries. The results indicate that, on average, adRNNCell is the best cell according to three accuracy measures, MAPE, MdAPE and RMSE. It also produces the least dispersed forecasts – see the lowest values of IqrAPE and StdPE. The second most accurate and precise cell is dRNNCell. The worst results are for GRU1.

Table 1. Forecasting quality metrics.

Cell type	MAPE	MdAPE	IqrAPE	RMSE	MPE	StdPE
GRU1	2.31	2.10	2.23	318.69	**-0.06**	3.86
GRU2	2.26	2.04	2.19	308.92	-0.15	3.78
LSTM1	2.25	2.03	2.18	307.09	-0.19	3.78
LSTM2	2.16	1.94	2.10	293.00	-0.10	3.60
dLSTM	2.19	1.97	2.12	297.58	-0.19	3.66
dRNNCell	2.15	1.93	2.09	292.60	-0.15	3.57
adRNNCell	**2.12**	**1.91**	**2.07**	**289.32**	-0.14	**3.52**

To confirm the performance of adRNNCell, we perform a pairwise one-sided Giacomini-White test (GM test) for conditional predictive ability [24] (we used the multivariate variant of the GW test implemented in https://github.com/jeslago/epftoolbox [25]). Fig. 7 shows the obtained p-values of this test. The closer the p-values are to zero the significantly more accurate the forecasts produced by the model on the X-axis are than the forecasts produced by the model on the Y-axis. The black color is for p-values larger than 0.10, indicating rejection of the hypothesis that the model on the X-axis is more accurate than the model on the Y-axis. Fig. 7 clearly shows that adRNNCell and dRNNCell performed best.

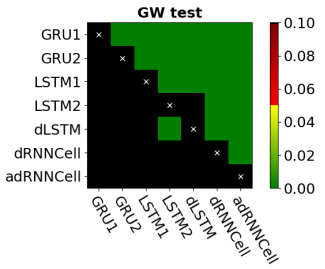


Fig. 7. Results of the Giacomini-White test.

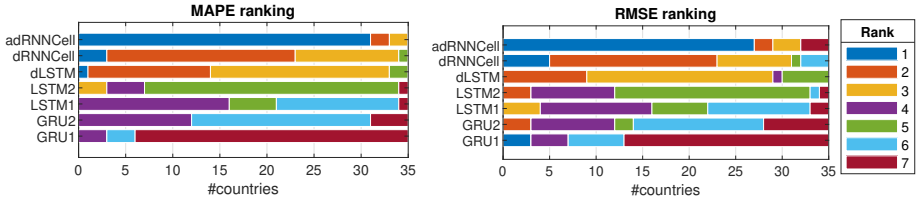


Fig. 8. Results of MAPE and RMSE rankings.

Fig. 8 shows rankings of the examined RNNs based on average errors for each country. Note the high position of adRNNCell. For 31 out of 35 countries this model gave the lowest MAPE, and for 27 countries it also gave the lowest RMSE. The second highest ranked model was dRNNCell.

In Table 1, we also show MPE, which is a measure of the forecast bias. Its negative values for all cases indicate over-prediction. The proposed model, thanks to the pinball-type loss function, can control the bias. For example, to reduce the bias for dRNNCell and adRNNCell we assumed $q^* = 0.485$. This resulted in a reduction of MPE to -0.04 without decreasing the forecast accuracy. So, it is possible to reduce the biases shown in Table 1 further, but we were trying to prevent over-tuning of the hyperparameters, so left them as they are reported.

Fig. 9 shows example forecasts of daily profiles for different days of the week. Note that forecasts generated by RNN with different cells do not differ much from each other. Fig. 9 also shows PIs for adRNNCell. To evaluate the accuracy of the PIs, we calculated the percentage of forecasts lying inside, above and below their PIs. The results are shown in Table 2. The predicted 90% PIs cover the forecasts most accurately in the case of GRU2. But note that our loss function (5) gives us the opportunity to tune further PIs. This can be performed by adjusting the quantiles determining the PI bounds, $\underline{q}$ and $\overline{q}$.

In Table 2, a Winkler score is also shown. For observations that fall within the PI, this score is simply the length of the PI, while for observations outside PI, the penalty applies, which is proportional to how far the observation is outside PI [26]. To bring the Winkler scores for different countries to a comparable level, we divide these scores by the mean loads of the corresponding countries in the test

period. Such unified Winkler scores are shown in Table 2. Note that adRNNCell has the lowest Winkler score and adRNNCell the second lowest.

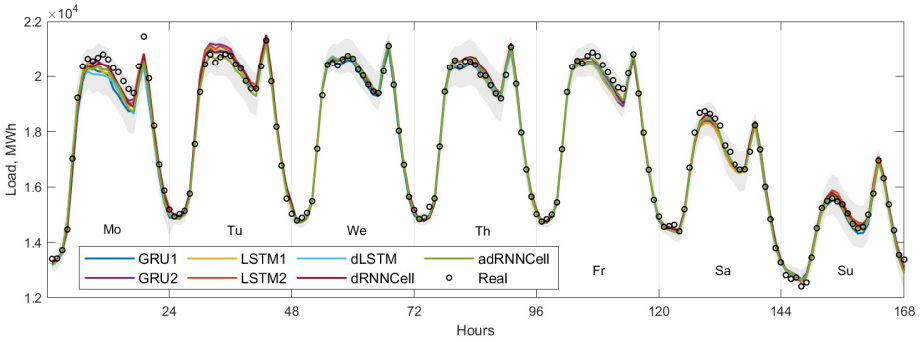


Fig. 9. Examples of the forecasts. 90% PIs for adRNNCell are shown as gray-shaded areas.

Table 2. Forecasting quality metrics.

Cell type	% in PI	% below PI	% above PI	Winkler score
GRU1	92.78±2.80	3.19±1.33	4.03±1.68	0.1524±0.2579
GRU2	**90.40±3.40**	**4.79±1.74**	**4.81±1.80**	0.1448±0.2710
LSTM1	89.16±4.07	5.30±2.05	5.53±2.21	0.1428±0.2760
LSTM2	88.52±3.88	5.53±1.96	5.96±2.04	0.1368±0.2741
dLSTM	88.84±3.74	5.59±1.96	5.57±1.94	0.1393±0.2737
dRNNCell	87.51±3.54	6.16±1.92	6.33±1.77	0.1363±0.2771
adRNNCell	88.41±3.14	5.68±1.67	5.91±1.62	**0.1332±0.2683**

Our research shows that adRNNCell is the best gated cell for forecasting time series with multiple seasonality. In [22] we compared RNN based on adRNNCells with a variety of forecasting models including statistical models (ARIMA, exponential smoothing, Prophet) and ML models (MLP, SVM, ANFIS, LSTM, GRNN, nonparametric models). This comparison clearly showed that the adRNNCell-based approach outperforms all its competitors in terms of accuracy.

6 Conclusion

In this study, we explore the potential of RNNs with different cells for forecasting time series with multiple seasonality. The best RNN solutions use dRNNCells

and adRNNCells, cells designed especially for such complex time series. They outperform classical GRU and LSTM cells as well as modified LSTM with dilation. adRNNCell, which is the most advanced cell with dilation and attention, combines two dRNNCells: one of which learns an attention vector while the other uses this vector to weight the inputs. The attention mechanism enables the cell to preprocess dynamically the sequence data while the delayed connections enable it to capture the long-term and seasonal dependencies in time series.

Apart from the dilation and attention mechanisms, the superior performance of the proposed RNN has its sources in the following mechanisms and procedures. First is the multilayer architecture, which is composed of several dilated recurrent layers stacked with hierarchical dilations to deal with multiple seasonality. Second is cross-learning on many time series, which enables RNN to capture the shared features of the individual series and helps to avoid over-fitting. Third is a time series representation using standardized weekly patterns as inputs and encoded daily patterns as outputs. The encoding variables are determined from the history, which enables decoding. Fourth is a composed asymmetrical loss function based on quantiles, which enables RNN to produce both point forecasts and PI and also to reduce the forecast bias.

In further research, we plan to enrich the input information with a learned context vector. This represents information extracted from other time series, which can help predict a given time series.

References

1. Box, G.E.P., Jenkins, G.M. and Reinsel, G.C.: Time series analysis: Forecasting and control. 3rd ed., Prentice Hall, New Jersey, 1994
2. Taylor, J.W.: Triple seasonal methods for short-term load forecasting. European Journal of Operational Research **204**, 139–152 (2010)
3. De Livera, A.M., Hyndman, R.J., Snyder, R.D.: Forecasting time series with complex seasonal patterns using exponential smoothing. Journal of the American Statistical Association **106**(496), 1513-1527 (2011)
4. Taylor, S.J., Letham, B.: Forecasting at scale. The American Statistician **72**, 37–45 (2018)
5. Høverstad, B.A., Tidemann, A., Langseth, H., Öztürk, P.: Short-term load forecasting with seasonal decomposition using evolution for parameter tuning. IEEE Transactions on Smart Grid **6**, 1904–1913 (2015)
6. Dudek, G.: Pattern-based local linear regression models for short-term load forecasting. Electric Power Systems Research **130**, 139–147 (2016)
7. Sharma, S., Majumdar, A., Elvira, V., Chouzenoux, E.: Blind Kalman filtering for short-term load forecasting. IEEE Transactions on Power Systems **35**, 4916–4919 (2020)
8. Benidis, K. et al.: Neural forecasting: Introduction and literature overview. ArXiv preprint arXiv:2004.10240 (2020)
9. Hewamalage, H., Bergmeir, C., Bandara, K.: Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting **37**, 388–427 (2021)
10. Yan, K. et al.: Multi-step short-term power consumption forecasting with a hybrid deep learning strategy. Energies **11**, 3089 (2018)

11. Bianchi, F.M., Maiorino, E., Kampffmeyer, M.C., Rizzi, A., Jenssen, R.: An overview and comparative analysis of recurrent neural networks for short term load forecasting. ArXiv preprint arXiv:1705.04378 (2017)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778 (2016)
13. Chang, S. et al.: Dilated recurrent neural networks. In: NIPS 2017 (2017).
14. Bandara, K., Bergmeir, C., Hewamalage, H.: LSTM-MSNet: Leveraging forecasts on sets of related time series with multiple seasonal patterns. IEEE Transactions on Neural Networks and Learning Systems **32**, 1586–1599 (2020)
15. Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting **36**, 75–85 (2020)
16. Dudek, G.: Neural networks for pattern-based short-term load forecasting: A comparative study. Neurocomputing **205**, 64–74 (2016)
17. Dudek, G., Pełka, P., Smyl, S.: 3ETS+ RD-LSTM: A new hybrid model for electrical energy consumption forecasting. In: International Conference on Neural Information Processing ICONIP, pp. 519–531. Springer (2020)
18. Smyl, S., Dudek, G., Pełka, P.: ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting. ArXiv preprint arXiv:2112.02663 (2021)
19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**, 1735–1780 (1997)
20. Cho, K. et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. ArXiv preprint arXiv:1406.1078 (2014)
21. Ben-Ari, I., Shwartz-Ziv, R.: Sequence modeling using a memory controller extension for LSTM. In: NIPS 2017 Time Series Workshop (2017)
22. Smyl, S., Dudek, G., Pełka, P.: ES-dRNN with dynamic attention for short-term load forecasting. ArXiv preprint arXiv:2203.00937 (2022)
23. Dudek, G., Pełka, P., Smyl, S.: A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting. IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2020.3046629 (2021)
24. Giacomini, R., White, H.: Tests of conditional predictive ability. Econometrica **74**, 1545–1578 (2006)
25. Lago, J., Marcjasz, G., De Schutter, B., Weron, R.: Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. Applied Energy **293**, 116983 (2021)
26. Hyndman, R.J., Athanasopoulos, G.: Forecasting: principles and practice. OTexts (2018). Accessed on February 2022