Randomized Neural Networks for Forecasting Time Series with Multiple Seasonality*

Grzegorz Dudek[0000−0002−2285−0327]

Electrical Engineering Faculty, Czestochowa University of Technology, Czestochowa, Poland
grzegorz.dudek@pcz.pl

Abstract. This work contributes to the development of neural forecasting models with novel randomization-based learning methods. These methods improve the fitting abilities of the neural model, in comparison to the standard method, by generating network parameters in accordance with the data and target function features. A pattern-based representation of time series makes the proposed approach useful for forecasting time series with multiple seasonality. In the simulation study, we evaluate the performance of the proposed models and find that they can compete in terms of forecasting accuracy with fully-trained networks. Extremely fast and easy training, simple architecture, ease of implementation, high accuracy as well as dealing with nonstationarity and multiple seasonality in time series make the proposed model very attractive for a wide range of complex time series forecasting problems.

Keywords: Multiple seasonality · Pattern representation of time series · Randomized neural networks · Short-term load forecasting · Time series forecasting.

1 Introduction

Time series (TS) expressing different phenomena and processes may include multiple seasonal cycles of different lengths. They can be observed in demand variations for various goods, weather conditions, customer numbers, stock market indicators or results of experimental research. Multiple seasonality in TS as well as nonstationarity, nonlinear trend and random fluctuations place high demands on forecasting models. The model should be flexible enough to capture these features without imposing too much computational burden. Over the years, many sophisticated forecasting models for TS with multiple seasonality have been proposed including statistical and machine learning (ML) ones.

One of the most commonly employed classical approaches, the autoregressive moving average model (ARMA), can be extended to multiple seasonal cycles by including additional seasonal factors [1]. Another popular statistical model,

* Supported by Grant 2017/27/B/ST6/01804 from the National Science Centre, Poland.

Holt–Winters exponential smoothing (ETS), was developed for forecasting TS data that exhibits both a trend and a seasonal variation. ETS was extended to incorporate a second and a third seasonal component in [2]. Both these models, ARMA and seasonal Holt–Winters model, have a significant weakness. They require the same cyclical behavior for each period. In [3], to cope with changing seasonal patterns, innovations state space models for ETS were proposed. The limitation of the model is that it can only be used for double seasonality where one seasonal length is a multiple of the other. A further extension of ETS was proposed in [4]. To deal with multiple seasonal periods, high-frequency, non-integer seasonality, and dual-calendar effects, it combines an ETS state space model with Fourier terms, a Box-Cox transformation and ARMA error correction.

As an alternative to statistical models, ML models have the ability to learn relationships between predictors and forecasted variables from historical data. One of the most popular in the well-stocked arsenal of ML methods are neural networks (NNs). A huge number of forecasting models based on different NN architectures have been proposed [5]. They deal with multiple seasonality differently, depending on the specific architectural features and the creativity of the authors. For example, the model that won the renowned M4 Makridakis competition combines ETS and recurrent NN (RNN) [6]. In this approach, ETS produces two seasonal components for TS deseasonalization and adaptive normalization during on-the-fly preprocessing, while RNN, i.e. long short term memory (LSTM), predicts the preprocessed TS.

Another example of using LSTM for forecasting TS with multiple seasonal patterns was proposed recently in [7]. To deal with multiple seasonal cycles, the model initially deseasonalizes TS using different strategies including Fourier transformation. RNNs, such as LSTM, gated recurrent units, and DeepAR [8], dominate today as NN architectures for TS forecasting thanks to their powerful ability to process sequential data and capture long-term dependencies. But other deep architectures are also useful for forecasting multiple seasonal TS. For example, N-Beats [9] was designed specifically for TS with multiple seasonality. It is distinguished by a specific architecture including backward and forward residual links and a very deep stack of fully-connected layers.

The above presented approaches to forecasting TS with multiple seasonal periods rely on incorporating into the model mechanisms which allow it to deal with seasonal components. This complicates the model and makes it difficult to train and optimize. An alternative approach is to simplify the forecasting problem by TS decomposition or preprocessing. In [10], TS with three seasonal cycles was represented by patterns expressing unified shapes of the basic cycle. This preprocessing simplified the relationship between TS elements, making decomposition unnecessary and removing the need to build a complex model. Instead, simple shallow NNs can be used [10] or nonparametric regression models [11]. Experimental research has confirmed that these models can compete in terms of accuracy with state-of-the-art deep learning models, like the winning M4 submission [12].

In this study, we use a pattern representation of TS to simplify the forecasting problem with multiple seasonality and propose randomization-based shallow NNs to solve it. Randomized learning was proposed as an alternative to gradient-based learning as the latter is known to be time-consuming, sensitive to the initial parameter values and unable to cope with the local minima of the loss function. In randomized learning, the parameters of the hidden nodes are selected randomly and stay fixed. Only the output weights are learned. This makes the optimization problem convex and allows us to solve it without tedious gradient descent backpropagation, but using a standard linear least-squares method instead [13]. This leads to a very fast training. The main problem in randomized learning is how to select the random parameters to ensure the high performance of the NN [14], [15]. In this study, to generate the random parameters we use three methods recently proposed in [16], [17]. These methods distribute the activation functions (sigmoids) of hidden nodes randomly in the input space and adjust their weights (or a weight interval) to the target function (TF) complexity using different approaches.

The main goal of this study is to show that randomization-based NNs can compete in terms of forecasting accuracy with fully-trained NNs. The contribution of this study can be summarized as follows:

1. A new forecasting model for TS with multiple seasonality based on randomized NNs is proposed. To deal with multiple seasonality and nonstationarity, the model applies pattern representation of TS in order to simplify the relationship between input and output data.
2. Three randomization-based methods are used to generate the NN hidden node parameters. They introduce steep fragments of sigmoids in the input space, which improves modeling of highly nonlinear TFs. A randomized approach leads to extremely fast and easy training, simple NN architecture and ease of implementation.
3. Numerical experiments on several real-world datasets demonstrate the efficiency of the proposed randomization-based models when compared to fully-trained NNs.

The remainder of this work is structured as follows. Section 2 presents the proposed forecasting model based on randomized NNs, and a TS representation using patterns of seasonal cycles and three methods of generating NN parameters are described. The performance of the proposed approach is evaluated in Section 3. Finally, Section 4 concludes the work.

2 Forecasting model

The proposed forecasting model is shown in Fig. 1. It is composed of encoder and decoder modules and a randomized feedforward NN (FNN). The model architecture, its specific features, and components are described below.

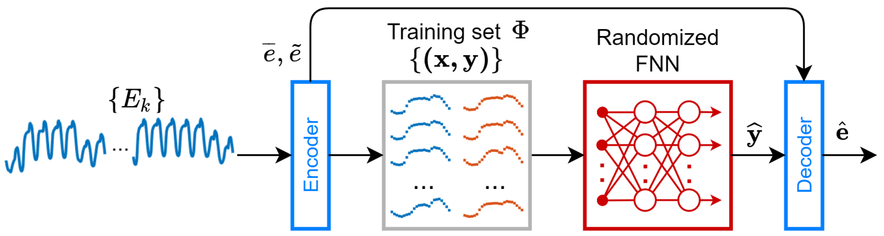

Fig. 1: Block diagram of the proposed forecasting model.

2.1 Encoder

The task of the encoder is to convert an original TS into unified input and output patterns of its seasonal cycles. To create input patterns, the TS expressing multiple seasonality, $\{E_k\}_{k=1}^K$, is divided into seasonal sequences of the shortest length. Let these sequences be expressed by vectors $\mathbf{e}_i = [E_{i,1}, E_{i,2}, \ldots, E_{i,n}]^T$, where n is the seasonal sequence length and $i = 1, 2, ..., K/n$ is the sequence number. These sequences are encoded in input patterns $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]^T$ as follows:

$$\mathbf{x}_i = \frac{\mathbf{e}_i - \overline{e}_i}{\widetilde{e}_i} \tag{1}$$

where $\overline{e}_i$ is a mean value of sequence $\mathbf{e}_i$, and $\widetilde{e}_i = \sqrt{\sum_{t=1}^n (E_{i,t} - \overline{e}_i)^2}$ is a measure of sequence $\mathbf{e}_i$ dispersion.

Note that the x-patterns are normalized versions of centered vectors $\mathbf{e}_i$. All x-patterns, representing successive seasonal sequences, have zero mean, the same variance and the same unity length. However, they differ in shape. Thus, the original seasonal sequences, which have a different mean value and dispersion, are unified. This is shown in Fig. 2 on the example of the hourly electricity demand TS expressing three seasonalities: daily, weekly, and yearly. Note that the x-patterns representing the daily cycles are all normalized and differ only in shape.

The output patterns $\mathbf{y}_i = [y_{i,1}, y_{i,2}, \ldots, y_{i,n}]^T$ represent the forecasted sequences $\mathbf{e}_{i+\tau} = [E_{i+\tau,1}, E_{i+\tau,2}, \ldots, E_{i+\tau,n}]^T$, where $\tau \geq 1$ is a forecast horizon. The y-patterns are determined as follows:

$$\mathbf{y}_i = \frac{\mathbf{e}_{i+\tau} - \overline{e}_i}{\widetilde{e}_i} \tag{2}$$

where $\overline{e}_i$ and $\widetilde{e}_i$ are the same as in (1).

Note that in (2), for the i-th output pattern, we use the same coding variables $\overline{e}_i$ and $\widetilde{e}_i$ as for the i-th input pattern. This is because the coding variables for the forecasted sequence, $\overline{e}_{i+\tau}$ and $\widetilde{e}_{i+\tau}$, are unknown for the future period. Using the coding variables determined from the previous period has consequences which are demonstrated in Fig. 2. Note that y-patterns in this figure reveal the weekly seasonality. The y-patterns of Mondays are much higher than the patterns of other days of the week because the Monday sequences are coded with the means of Sunday sequences which are much lower than the means of Monday sequences. For similar reasons, y-patterns for Saturdays and Sundays are lower than y-patterns for the other days of the week. Thus, the y-patterns are not unified globally but are unified in groups composed of the same days of the week. For this reason, we construct forecasting models that learn from data representing the same days of the week. For example, when we train the model to forecast the daily sequence for Monday, the training set for it, $\Phi = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, is composed of the y-patterns representing all Mondays from history and corresponding x-patterns representing the previous days (depending on the forecast horizon; Sundays for $\tau = 1$).

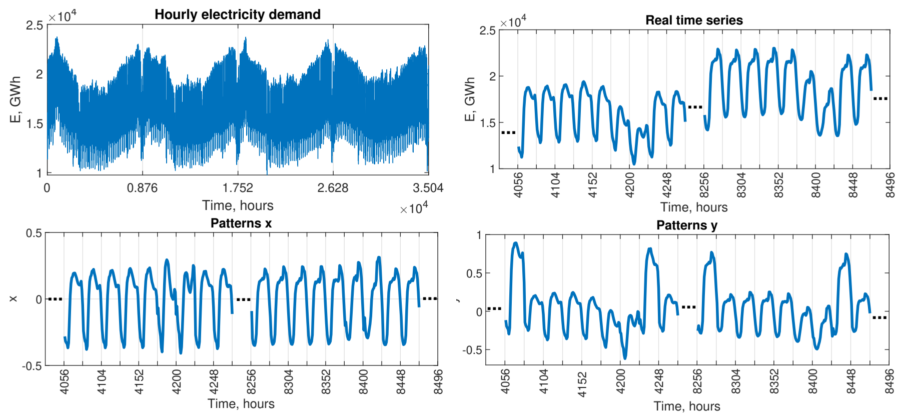

Fig. 2: Real hourly electricity demand TS and its x- and y-patterns.

2.2 Decoder

The decoder converts a forecasted output pattern into a TS seasonal cycle. The output pattern predicted by randomized FNN is decoded using the coding variables of the input query pattern, $\mathbf{x}$, using transformed equation (2):

$$\widehat{\mathbf{e}} = \widehat{\mathbf{y}}\widetilde{e} + \overline{e} \tag{3}$$

where $\widehat{\mathbf{e}}$ is the forecasted seasonal sequence, $\widehat{\mathbf{y}}$ is the forecasted output pattern, $\widetilde{e}$ and $\overline{e}$ are the coding variables determined from the TS sequence encoded in query pattern $\mathbf{x}$.

2.3 Randomized FNN

The randomized FNN is composed of n inputs, one hidden layer with m nonlinear nodes, and n outputs. Logistic sigmoid activation functions are employed for hidden nodes. The training set is $\Phi = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}, \mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^n$. The randomized learning algorithm consists of three steps [18].

1. Randomly generate hidden node parameters, i.e. weights $\mathbf{a}_j = [a_{j,1}, a_{j,2}, \dots, a_{j,n}]^T$ and biases $b_j, j = 1, 2, \dots, m$, according to any continuous sampling distribution.
2. Calculate the hidden layer output matrix:
$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} \tag{4}$$
where $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_m(\mathbf{x})]$ is a nonlinear feature mapping from n-dimensional input space to m-dimensional feature space, and $h_j(\mathbf{x})$ is an activation function of the j-th node (a sigmoid in our case).
3. Calculate the output weights:
$$\boldsymbol{\beta} = \mathbf{H}^{+}\mathbf{Y} \tag{5}$$
where $\boldsymbol{\beta} \in \mathbb{R}^{m \times n}$ is a matrix of output weights, $\mathbf{Y} \in \mathbb{R}^{N \times n}$ is a matrix of target output patterns, and $\mathbf{H}^{+} \in \mathbb{R}^{m \times N}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

Typically, the hidden node weights and biases are i.i.d random variables both generated from the same symmetrical interval $a_{j,i}, b_j \sim U(-u, u)$. It was pointed out in [18] and [16] that as the weights and biases have different functions they should be selected separately. The weights decide about the sigmoid slopes and should reflect the TF complexity, while the biases decide about the sigmoid shift and should ensure the placement of the most nonlinear sigmoid fragments, i.e. the fragments around the sigmoid inflection points, into the input hypercube. These fragments, unlike saturation fragments, are most useful for modeling TF fluctuations.

Recently, to improve the performance of randomized FNNs, several new methods of generating the hidden node parameters have been proposed. Among them is the random a method (RaM) which was proposed in [16]. In the first step, this method randomly selects weights from the interval whose bounds u are adjusted to the TF complexity, $a_{j,i} \sim U(-u, u)$. Then, to ensure the introduction of the sigmoid inflection points into the input hypercube, the biases are calculated from:

$$b_j = -\mathbf{a}_j^T \mathbf{x}_j^* \tag{6}$$

where $\mathbf{x}_j^*$ is one of the training x-patterns selected for the j-th hidden node at random.

The second method proposed in [16], called the random α method (RαM), instead of generating weights, generates the slope angles of sigmoids. This changes the distribution of weights, which typically is a uniform one. This new distribution ensures that the slope angles of sigmoids are uniformly distributed, and so improves results by preventing overfitting, especially for highly nonlinear TFs. This method, in the first step, selects randomly the slope angles of the sigmoids, $\alpha_{j,i} \sim U(\alpha_{\min}, \alpha_{\max})$. Then, the the weights are calculated from:

$$a_{j,i} = 4 \tan \alpha_{j,i} \tag{7}$$

Finally, the biases are determined from (6). To simplify the optimization process, the lower bound for the angles, $\alpha_{\min}$, can be set as 0°. In such a case only one parameter decides about the model flexibility, i.e. $\alpha_{\max} \in (0°, 90°)$. This is what we used in our simulation study.

To improve further FNN randomized learning, a data-driven method (DDM) was proposed in [17]. This method introduces the sigmoids into randomly selected regions of the input space and adjusts the sigmoid slopes to the TF slopes in these regions. As a result, the sigmoids mimic the TF locally, and their linear combination approximates smoothly the entire TF. In the first step, DDM selects the input space regions by selecting randomly the set of training points, $\{\mathbf{x}_j^*\}_{j=1}^m$. Then, the hyperplanes are fitted to the TF locally in the neighbourhoods of all points $\mathbf{x}_j^*$. The neighborhood of point $\mathbf{x}_j^*$, $\Psi(\mathbf{x}_j^*)$, contains this point and its k nearest neighbors in Φ. The weights are determined based on the hyperplane coefficients from:

$$a_{j,i} = 4a'_{j,i} \tag{8}$$

where $a'_{j,i}$ are the coefficients of the hyperplane fitted to neighbourhood $\Psi(\mathbf{x}_j^*)$.

The hidden node biases are calculated from (6).

Note that the biases in the above-described approaches are determined based on the weights selected first and the data points. Unlike in the standard approach, they are not chosen randomly from the same interval as the weights. Randomized FNN has two hyperparameters to adjust: number of hidden nodes m, and the smoothing parameter, i.e. u, $\alpha_{\max}$ or k, depending on the method of generating parameters chosen. These hyperparameters decide about the fitting performance of the model and its bias-variance tradeoff. Their optimal values should be selected by cross-validation for a given forecasting problem.

3 Simulation Study

In this section, we apply the proposed randomization-based neural models to forecasting hourly TS with three seasonalities: yearly, weekly and daily. These TS express electricity demand for four European countries: Poland (PL), Great Britain (GB), France (FR) and Germany (DE). We use real-world data collected from www.entsoe.eu. The data period covers the 4 years from 2012 to 2015. Atypical days such as public holidays were excluded from these data (between 10 and 20 days a year). The forecast horizon τ is one day, i.e. 24 hours. We forecast the daily load profile for each day of 2015. For each forecasted day, a new training set is created and a new randomized model is optimized and trained. The results presented below are averaged over 100 independent training sessions.

The hyperparameters of randomized FNNs were selected using grid search and 5-fold cross-validation. The number of hidden nodes was selected from the set $\{5, 10, ..., 50\}$. The bounds for weights in RaM were selected from $\{0.02, 0.04, ..., 0.2, 0.4, ..., 1\}$. The $\alpha_{\max}$ in RαM was selected from $\{2°, 4°, ..., 40°, 45°, ..., 90°\}$. The number of nearest neighbors in DDM was selected from $\{25, 27, ..., 69\}$.

For comparison, we applied a multilayer perceptron (MLP) for the same forecasting problems. MLP was composed of a single hidden layer with m sigmoid nodes whose number was selected using 5-fold cross-validation from the set $\{2, 4, ..., 24\}$. MLP was trained using Levenberg-Marquardt backpropagation with early stopping to avoid overtraining (20% of training samples were used as validation samples).

Forecasting quality metrics for the test data are presented in Table 1. They include: mean absolute percentage error (MAPE), median of APE, root mean square error (RMSE), mean percentage error (MPE), and standard deviation of percentage error (PE) as a measure of the forecast dispersion.

Table 1: Forecasting results.

		RaM	RαM	DDM	MLP
PL data	MAPE	1.32	1.32	1.35	1.37
	Median(APE)	0.93	0.94	0.94	0.96
	RMSE	358.86	364.13	380.77	374.86
	MPE	0.40	0.39	0.39	0.26
	Std(PE)	1.94	1.98	2.09	2.07
GB data	MAPE	2.61	2.62	2.80	2.93
	Median(APE)	1.88	1.90	1.99	2.17
	RMSE	1187.60	1184.58	1382.97	1314.78
	MPE	-0.61	-0.61	-0.58	-0.60
	Std(PE)	3.57	3.56	4.16	3.99
FR data	MAPE	1.67	1.69	1.81	1.87
	Median(APE)	1.15	1.16	1.25	1.31
	RMSE	1422.60	1433.90	1530.15	1565.70
	MPE	-0.42	-0.39	-0.45	-0.39
	Std(PE)	2.60	2.61	2.78	2.85
DE data	MAPE	1.38	1.39	1.43	1.58
	Median(APE)	0.96	0.98	0.99	1.09
	RMSE	1281.14	1242.36	1333.79	1452.54
	MPE	0.14	0.14	0.10	0.04
	Std(PE)	2.22	2.13	2.34	2.50

More detailed results, i.e. distributions of APE, are shown in Fig. 3. Based on APE, we performed a Wilcoxon signed-rank test with $\alpha = 0.05$ to indicate the most accurate models. Fig. 4 depicts pairwise comparisons of the models. The arrow lying at the intersection of the two models indicates which of them gave the significantly lower error. A lack of an arrow means that both models gave statistically indistinguishable errors.

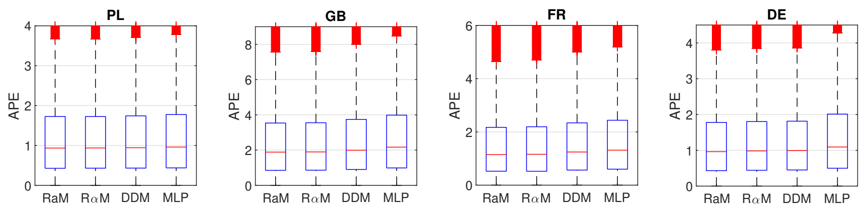

Fig. 3: Boxplots of APE.

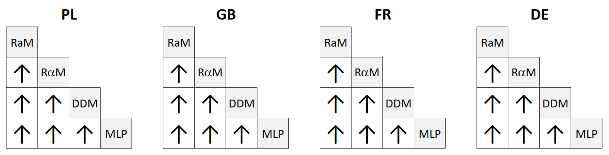

Fig. 4: Results of the Wilcoxon signed-rank test for APE.

As can be seen from Table 1 and Fig. 4, the randomization-based FNNs gave significantly lower errors than fully-trained MLP for each dataset. According to the Wilcoxon test, RαM outperformed the other approaches.

MPE shown in Table 1 allows us to asses the bias of the forecasts produced by different models. A positive value of MPE indicates underprediction, while a negative value indicates overprediction. As can be seen from Table 1, for PL and DE data the bias was positive, whilst for GB and FR data it was negative. The forecasts produced by MLP for PL and DE were less biased than the forecast produced by randomized FNNs.

Fig. 5 presents examples of forecasts of the daily load profiles produced by the examined models. Note that the proposed models generate multi-output response, maintaining the relationships between the output variables (y-pattern components). In the case of single-output models, these relationships are ignored because the variables are predicted independently. This may cause a lack of smoothness in the forecasted curve (zigzag effect; see for example [19]).

Fig. 6 shows the optimal numbers of hidden nodes selected in the cross-validation procedure. Obviously, the number of hidden nodes is dependent on TF complexity. The forecasting problem for PL required the greatest number of

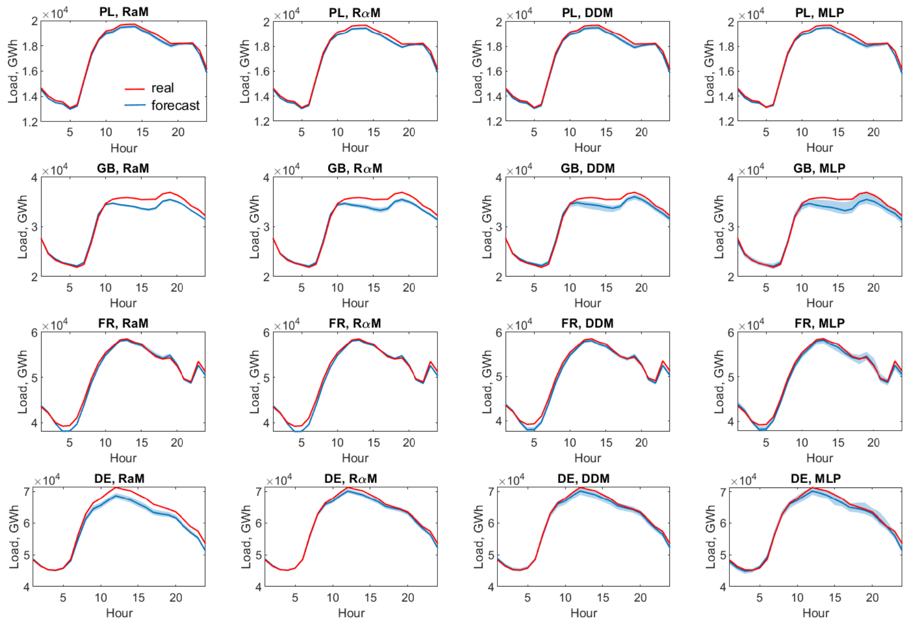

Fig. 5: Examples of forecasts (shaded regions are 5th and 95th percentiles, measured over 100 trials).

nodes for randomized FNNs, around 30, regardless of the learning method. MLP for PL needed many fewer nodes, 12 on average. Other forecasting problems were solved by randomized FNNs with fewer hidden nodes, from 20 to 30 on average. For these problems, the difference in the number of nodes between MLP and randomized FNNs was not as large as for PL data. The relatively small number of hidden nodes in randomized FNNs (note that randomized learning usually requires hundreds or even thousands of nodes) results from TS representation by unified patterns and the decomposition of the forecasting problem (a separate model for each forecasting task, i.e. every day in 2015, trained on the selected patterns).

The optimal values of smoothing parameters for the randomized learning methods are depicted in Fig. 7. As can be seen from this figure, the optimal value of the bound for weights in RaM varies from 0.2 for FR to 0.7 for GB on average, which correspond to sigmoid slope angles from around 3° to 10° (see [16]). The optimal value of the bound for slope angle in RαM varies from 12° for FR to 32° for DE on average. Note also the high value of k in DDM (from 49 for PL to 65 for FR on average) in relation to the number of training points, which ranged from 150 to 200. Thus, for our forecasting problems we can expect flat TFs without fluctuations. Such TFs can be modeled using RaM. Its competitors, RαM and DDM, reveal their strengths in modeling highly nonlinear TFs with fluctuations (see [16], [17]).

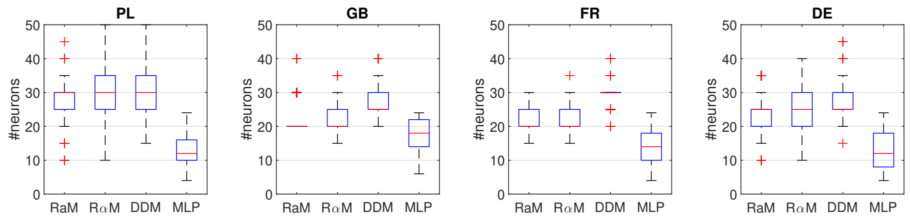

Fig. 6: Boxplots of the optimal number of hidden nodes.

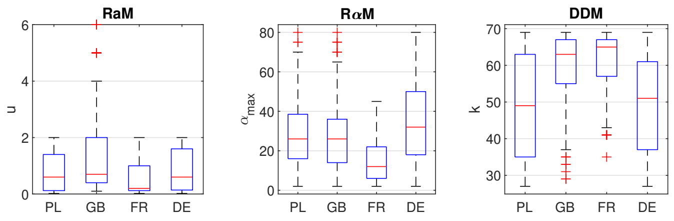

Fig. 7: Boxplots of the optimal smoothing parameters.

4 Conclusion

Forecasting TS with multiple seasonality is a challenging problem, which we propose to solve with randomized FNNs. Unlike fully-trained FNNs, randomized FNNs learn extremely fast and are easy to implement. The simulation study showed that their forecasting accuracy is comparable to the accuracy of fully-trained NNs. To deal with nonstationary TS with multiple seasonal periods, the proposed approach employs a pattern representation of the TS. This representation simplifies the relationship between input and output data and makes the problem easier to solve using simple regression models.

The effectiveness of the randomized FNNs in modeling nonlinear target functions was achieved due to the application of new methods of generating hidden node parameters. These methods, using different approaches, introduce the steepest fragments of sigmoids, which are most useful for modeling TF fluctuations, into the input hypercube and adjust their slopes to TF complexity. This makes the model more flexible, more data-dependent, and more dependent on the complexity of the solved forecasting problem.

In a future study, we plan to introduce an attention mechanism into our randomization-based forecasting models to select training data and develop an ensemble approach for these models.

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. Gould, P.G., Koehler, A.B., Ord, J.K., Snyder, R.D., Hyndman, R.J. and Vahid-Araghi, F.: Forecasting time-series with multiple seasonal patterns. European Journal of Operational Research **191**, 207–222 (2008)
4. De Livera, A.M., Hyndman, R.J. and Snyder, R.D.: Forecasting time series with complex seasonal patterns using exponential smoothing. Journal of the American Statistical Association **106**(496), 1513–1527 (2011)
5. Benidis, K., Rangapuram, S.S., Flunkert, V., Wang, B., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., Callot L. and Januschowski, T.: Neural forecasting: Introduction and literature overview. arXiv:2004.10240 (2020)
6. Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting **36**(1), 75–85 (2020)
7. Bandara, K., Bergmeir C. and 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**(4), 1586–1599 (2021)
8. Salinas, D., Flunkert, V., Gasthaus, J. and Januschowski, J.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181–1191 (2020)
9. Oreshkin, B.N., Carpov, D., Chapados, N. and Bengio, Y.: N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. 8th International Conference on Learning Representations, ICLR, (2020).
10. Dudek, G.: Neural networks for pattern-based short-term load forecasting: A comparative study. Neurocomputing **205**, 64–74 (2016)
11. Dudek, G.: Pattern similarity-based methods for short-term load forecasting – Part 1: Principles. Applied Soft Computing **37**, 277–287 (2015).
12. Dudek, G., Pełka, P. and Smyl, S.: A hybrid residual dilated LSTM and exponential smoothing model for mid-term electric load forecasting. IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2020.3046629
13. Principe, J., Chen, B.: Universal approximation with convex optimization: Gimmick or reality? IEEE Computational Intelligence Magazine **10**(2), 68–77 (2015)
14. Cao, W., Wang, X., Ming, Z., Gao, J.: A review on neural networks with random weights. Neurocomputing **275**, 278–287 (2018)
15. Zhang, L., Suganthan, P.: A survey of randomized algorithms for training neural networks. Information Sciences **364–365**, 146–155 (2016)
16. Dudek, G.: Generating random parameters in feedforward neural networks with random hidden nodes: Drawbacks of the standard method and how to improve it. In: Neural Information Processing. ICONIP 2020. CCIS, vol. 1333, pp. 598–606, Springer, Cham (2020).
17. Dudek, G.:Data-driven randomized learning of feedforward neural networks, 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, pp. 1–8 (2020).
18. Dudek, G.: Generating random weights and biases in feedforward neural networks with random hidden nodes. Information Sciences, **481**, 33–56 (2019)
19. Dudek G.: Multivariate regression tree for pattern-based forecasting time series with multiple seasonal cycles. In: Information Systems Architecture and Technology: Proc. 38th International Conference on Information Systems Architecture and Technology – ISAT 2017. AISC, vol. 655. pp. 85–94, Springer, Cham (2018).