

CZESTOCHOWA UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

PhD Thesis

Discipline of science: Information and Communication Technology

Field of science: Engineering and Technology

Methods for improving performance of neural network-based forecasting models

Sławomir Smyl

Supervisor: professor Grzegorz Dudek, PhD, DSc, Eng

Auxiliary supervisor: Paweł Pełka, PhD, Eng

Częstochowa 2026

Contents

Abstract.....	3
Abstract in Polish	4
List of papers included in the dissertation.....	5
1. Introduction.....	7
2. Achievements and other publications	9
3. Literature overview	11
4. Paper [D1]: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting.....	21
5. Paper [D2]: Machine learning methods for GEFCom2017 probabilistic load forecasting	27
6. Paper [D3]: A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting.....	29
7. Paper [D4]: 3ETS+RD-LSTM: A new hybrid model for electrical energy consumption forecasting.....	31
8. Paper [D5]: ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting.....	33
9. Paper [D6]: ES-dRNN with dynamic attention for short-term load forecasting.....	38
10. Paper [D7]: Contextually enhanced ES-dRNN with dynamic attention for short-term load forecasting.....	39
11. Paper [D8]: Any-quantile probabilistic forecasting of short-term electricity demand: Fusing uncertainties from diverse sources	42
12. Summary	46
References	48
Papers	
[D1] A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting	53
[D2] Machine learning methods for GEFCom2017 probabilistic load forecasting.....	64

[D3]	A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting	72
[D4]	3ETS+RD-LSTM: A new hybrid model for electrical energy consumption forecasting	85
[D5]	ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting	98
[D6]	ES-dRNN with dynamic attention for short-term load forecasting	111
[D7]	Contextually enhanced ES-dRNN with dynamic attention for short-term load forecasting	119
[D8]	Any-quantile probabilistic forecasting of short-term electricity demand: Fusing uncertainties from diverse sources	132

Abstract

Over the past decade, neural networks have achieved remarkable success in many fields, such as image, text, and speech processing. In recent years, their use in time series forecasting has also become increasingly common. This, however, requires a specialized approach, as data preparation, system architecture, and even training procedures are, or at least should be, different from those used in other application domains.

This dissertation is based on a series of recent papers in which the author proposed various improvements to neural networks for time series forecasting. The most important of these include: on-the-fly (time-dynamic) data normalization; hybrid architectures that integrate exponential smoothing into neural network formulations; the use of global context with the ability to adapt to individual time series; training quantiles that differ slightly from the target quantiles; starting temporal steps at randomly selected points in order to delay overfitting; warmup procedures for recurrent networks; global models with the ability to incorporate local (series-specific) parameters; and dynamic ensembles of neural networks.

Keywords

Time series forecasting, neural networks, recurrent neural networks, hybrid models, exponential smoothing, global models, probabilistic forecasting, quantile forecasting, model ensembling, data preprocessing

Metody poprawy wydajności modeli prognostycznych opartych na sieciach neuronowych

W ciągu ostatniej dekady sieci neuronowe osiągnęły duże sukcesy w wielu dziedzinach, takich jak przetwarzanie obrazów, tekstu i dźwięku. W ostatnich latach obserwuje się również systematyczny wzrost zainteresowania ich zastosowaniem w prognozowaniu szeregów czasowych. Wymaga to jednak specjalnego podejścia, ponieważ przygotowanie danych, architektura systemu, a nawet procedury trenujące są, a raczej powinny być, odmienne od tych używanych w innych dziedzinach.

Praca bazuje na artykułach z ostatnich lat, w których autor proponował usprawnienia sieci neuronowych do prognozowania szeregów czasowych. Najważniejsze z tych usprawnień to: bieżąca (dynamiczna w czasie) normalizacja danych, architektura hybrydowa (wplecenie wygładzania wykładniczego do formuł sieci neuronowych), globalny kontekst, ale z możliwością dopasowania do każdego szeregu, kwantyle treningowe (które są nieco różne od docelowych kwantyli), rozpoczynanie kroków czasowych w losowo wybranych momentach (aby opóźnić moment przetrenowania), „rozgrzewka” sieci rekurencyjnych, globalny model, ale z możliwością lokalnych (zależnych od szeregu czasowego) parametrów, dynamiczne zespoły sieci neuronowych.

Słowa kluczowe

Prognozowanie szeregów czasowych, sieci neuronowe, rekurencyjne sieci neuronowe, modele hybrydowe, wygładzanie wykładnicze, modele globalne, prognozowanie probabilistyczne, prognozowanie kwantylowe, zespoły modeli, przetwarzanie wstępne danych

List of papers included in the dissertation

- [D1] Smyl S.: *A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting*. International Journal of Forecasting, vol. 36(1), pp. 75-85, 2020.
<https://doi.org/10.1016/j.ijforecast.2019.03.017>
IF = 3.779, 140 p.
Contribution: Ideation and coding, writing the paper.
- [D2] Smyl S., Hua G.: *Machine learning methods for GEFCom2017 probabilistic load forecasting*. International Journal of Forecasting, vol. 35(4), pp. 1424-1431, 2019.
<https://doi.org/10.1016/j.ijforecast.2019.02.002>
IF = 3.779, 140 p.
Contribution: Development of NN and QRF models, writing most of the paper.
- [D3] Dudek G., Pełka P., 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, vol. 33, no. 7, pp. 2879-2891, 2022.
<https://doi.org/10.1109/TNNLS.2020.3046629>
IF = 10.451, 200 p.
Contribution: Development of the algorithmic and experimental parts (development, implementation, optimization, training, and testing of the main model: a hybrid residual dilated LSTM and exponential smoothing model), concluding analysis, and drafting the paper.
- [D4] Dudek G., Pełka P., Smyl S.: 3ETS+RD-LSTM: *A new hybrid model for electrical energy consumption forecasting*. In: Yang H. et al. (eds) Neural Information Processing. ICONIP 2020, LNCS, vol. 12534, pp. 519-531, Springer, Cham, 2020.
https://doi.org/10.1007/978-3-030-63836-8_43
Ranked A conference in CORE ranking, 140 p.
Contribution: I provided a code that served as a starting point for this paper and reviewed it.
- [D5] Smyl S., Dudek G., Pełka P.: *ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting*. IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 8, pp. 11346-11358, 2024, doi:
<https://doi.org/10.1109/TNNLS.2023.3259149>
IF = 14.255, 200 p.
Contribution: The main model development, testing, optimization, and writing some parts of the paper (primarily the model description).

- [D6] Smyl S., Dudek G., Pelka P.: *ES-dRNN with dynamic attention for short-term load forecasting*. In 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 2022, doi: <https://doi.org/10.1109/IJCNN55064.2022.9889791>
Ranked A conference in CORE ranking, 140 p.
Contribution: The main model development, testing, optimization, and writing some parts of the paper (contributing to the model description).
- [D7] Smyl S., Dudek G., Pelka P.: *Contextually enhanced ES-dRNN with dynamic attention for short-term load forecasting*. Neural Networks, vol. 169, pp. 660-672, 2024, doi: <https://doi.org/10.1016/j.neunet.2023.11.017>.
IF = 7.8, 200 p.
Contribution: The main model development, testing, optimization, and writing some parts of the paper (primarily the model description).
- [D8] Smyl S., Oreshkin B., Pelka P., Dudek G.: *Any-quantile probabilistic forecasting of short-term electricity demand: Fusing uncertainties from diverse sources*. Information Fusion, vol.127, part A, pp. 103637, 2026, doi: <https://doi.org/10.1016/j.inffus.2025.103637>
IF = 15.5, 200 p.
Contribution: The main model development, testing, optimization, and writing some parts of the paper (primarily the model description).

1. Introduction

Forecasting or future prediction has been practiced since the dawn of civilization, using various methods, including analysis, such as observing what other people are likely to do, heuristics, like following recent trends, identifying patterns, and the occult, among others. Since the 19th century, probability analysis and statistical tools, such as linear regression, have been employed. In the second half of the 20th century, time series-specific algorithms, such as Exponential Smoothing, followed by precursors of ARIMA, emerged. They became more sophisticated, were formulated in rigorous mathematical terms, and were ultimately implemented in widely available packages by the end of the 20th century.

This coincided with interest in using Machine Learning (ML) for forecasting. However, generally speaking, these early attempts failed, e.g., during the M3 Forecasting Competition, ML methods performed substantially worse than the established (e.g., ARIMA) or new statistical methods (e.g., Theta), and a consensus held, until at least 2016-2018, that ML methods are unable to best the statistical techniques (unless working with series with strong regressors, e.g., promotions, etc.). I believe that the primary reason for these early failures was the incorrect approach of trying to fit a separate ML model for each series. This approach worked well for the statistical models, primarily due to their relative simplicity, but it was the wrong approach when using ML models with thousands or millions of parameters. At the same time, building a global model across multiple series with varying amplitudes and lengths was not possible or unsuccessful without extensive preprocessing, and practitioners were not accustomed to it.

Neural networks tend to be even more demanding of appropriate preprocessing than other ML methods. Additionally, their architecture, when borrowed from different disciplines such as sound and image processing, requires adjustment.

Although forecasts with Prediction Intervals (PIs) and, more broadly, density (also called probabilistic) forecasts have been known for a long time, they have only recently gained wider use among practitioners. Neural networks are particularly suitable for density forecasting, and there are several ways in which probabilistic forecasting can be achieved with them.

1.1. Research thesis and aim

The research thesis of this dissertation is as follows:

The performance of the neural network-based forecasting models can be improved through appropriate preprocessing (i.e., data representation), architectures aligned with the task, suitable training procedures, and ensembling.

Accordingly, the primary objective of this work is to develop and empirically validate methods that enhance the accuracy, stability, and practical usefulness of neural network-based forecasting models. This objective is pursued through the following specific goals:

- identifying the key components of the forecasting process – data preparation, model architecture, training methodology, and forecast aggregation – that have the greatest impact on predictive performance;
- designing universal and portable solutions that can be applied across a wide range of time series forecasting tasks and datasets;
- developing hybrid ES–RNN models that combine the strengths of classical statistical methods with those of neural networks;
- improving the quality of probabilistic forecasts, including the construction of well-calibrated predictive distributions;
- demonstrating that appropriately designed neural network models can outperform classical statistical forecasting methods, as evidenced by results obtained in international forecasting competitions.

1.2. Contribution and novelty

1. Preprocessing
 - a. Adaptive normalization
 - b. Escape seasonality trick
 - c. Hybrid Exponential Smoothing-RNN on-the-fly preprocessing
 - d. Adjusting Exponential Smoothing parameters dynamically
 - e. Enriching input for each series with a global context.
2. Training
 - a. Level wiggleness penalty
 - b. Training quantiles
 - c. Random starts, warmup
 - d. Epoch is defined as a sublinear function of the number of updates
 - e. Sampling quantile level from the Beta distribution
3. Architecture
 - a. Global model
 - b. New cells designed to work in a dilated RNN
 - c. Global and per-series parameters
 - d. Ensembling, specialists
 - e. Dynamic teams

1.3. Scope and layout

First, I summarize my achievements in the field of time series forecasting. A literature review follows this. I will summarize recent and essential papers. Then, each of the following eight chapters summarizes one of my papers. Looking back, I try to extract elements that worked well not only in a particular paper but also became my standard toolkit, proven in many other projects; I also mention weaknesses and dead ends. The sequence is broadly chronological, but I present my M4 forecasting competition paper first, as it is somewhat foundational in nature. The work ends with a conclusion chapter.

2. Achievements and other publications

I won the Computational Intelligence in Forecasting Competition (CIF2015) (Stepnicka & Burda, 2015) with an ensemble of an RNN-based system and a novel Bayesian Exponential Smoothing model fitted with Markov Chain Monte Carlo. This was the first time an expertly prepared ensemble of statistical models was beaten by an ML system in a forecasting competition.

My colleague and I got second place in the first round of the Global Energy Forecasting Competition 2017 (Hong, Xie, & Black, 2019). We used a mix of neural networks and tree-based algorithms. We achieved fourth place in the second round.

I won M4 (fourth edition) of Makridakis Forecasting Competitions - the most important and very influential sequence of competitions in the forecasting community (Hyndman, 2018). I won both accuracy (point forecast) and prediction intervals tracks. A new trend of using ML, and in particular NN methods, has started.

This thesis concentrates on neural networks. However, I also have developed a family of novel Bayesian Exponential Smoothing models fitted with Markov Chain Monte Carlo, called Local and Global Trend (LGT) models, (Smyl & Zhang, 2015) (Smyl, et al., 2025) (Smyl & Kuber, 2016) that were later sped up by a custom MCMC sampler (Long, Schmidt, Bergmeir, & Smyl, 2025). The models are still likely the best on the M3 dataset.

I have been presenting at the International Symposium on Forecasting every year since 2015, sometimes as an invited speaker, as in 2023 (Bergmeir & Smyl, 2023).

Below, I provide a summary and brief description of my publications and contributions (excluding [D1–D8]):

(Loboz, Smyl, & Nath, 2010) – this paper describes a large-scale system for storing and retrieving operating system counters. I contributed ideas and coded some relatively small parts of the system.

(Loboz, Jia, & Smyl, 2012) – a patent based on the above system.

(Smyl & Zhang, 2015) – a talk at the International Symposium on Forecasting describing my Bayesian LGT models. I created the idea, wrote the code, and the presentation.

(Smyl & Kuber, 2016) – a talk at the International Symposium on Forecasting on a system that used the LGT models for data augmentation for a neural network-based forecaster. I created the idea, wrote the code, and prepared the presentation.

(Laptev, Yosinski, Li, & Smyl, 2017) – an International Conference on Machine Learning (ICML) paper on a neural network-based forecasting system we developed at Uber. My contribution: discussions and feedback.

(Smyl, 2017) – a talk at the International Symposium on Forecasting on an ensemble of specialized neural networks. It is described in Chapter 4. I created the idea, wrote the code, and prepared the presentation.

(Smyl, Ranganathan, & Pasqua, 2018) – a blog post about my winning in the M4 Forecasting Competition.

- (Ng, Wang, Chen, Yang, & Smyl, 2020) – a paper describing a family of Bayesian forecasting models based on my LGT models. My contribution: discussions and feedback.
- (Bandara, Bergmeir, & Smyl, 2020) – a paper published in Expert Systems with Applications on grouping series into clusters before training and forecasting. I provided the original neural network code and participated in discussions.
- (Godaheewa, Bandara, Webb, Smyl, & Bergmeir, 2021) – a paper published in Knowledge-Based Systems that explored how various clustering approaches can improve neural network-based forecasting. I provided the original neural network code and participated in discussions.
- (Pełka, Dudek, & Smyl, 2022) – a paper at the Polish Conference on Artificial Intelligence on forecasting short-term electric load. I provided some of the neural network code.
- (Dudek, Smyl, & Pełka, 2023) – this paper compares various RNN cells used for time series forecasting. I provided some of the neural network code.
- (Smyl, Dudek, & Pełka, 2023) – International Conference on Computational Science paper on forecasting cryptocurrencies. I wrote the code.
- (Bergmeir & Smyl, 2023) – an invited talk at the International Symposium on Forecasting on dealing with high-frequency series by forecasting their forecast summaries. I wrote my part of the code and the talk.
- (Chen, Ng, Smyl, & Steininger, 2024) – a paper summarizing one of the major projects I lead at Uber – forecasting Customer Lifetime Value. I created the original code and wrote some parts of the paper.
- (Sriramulu, Bergmeir, & Smyl, 2024) – a paper comparing Graph Neural Networks and my RNN networks. I wrote the original code.
- (Long, Schmidt, Bergmeir, & Smyl, 2025) – a paper published in Statistics and Computing on improvements to my LGT models. My contribution: some discussions.
- (Smyl, et al., 2025) – a paper published in the International Journal of Forecasting describing my LGT models. I wrote the original code and part of the paper.

3. Literature overview

This review is based primarily on the following papers (Benidis, et al., 2022), (Kim, Kim, Kim, & Lee, 2025), (Hewamalage, Bergmeir, & Bandara, 2021), (Corradini, et al., 2025), and (Bergmeir, 2024), interspersed with my comments.

The first paper provides a technical overview of the state of forecasting with neural networks as of 2022. The second paper provides a broader historical overview and, being more recent (July 2025), better reflects current research. The third paper is a review of the most important, in my opinion, type of NNs used for forecasting – Recurrent Neural Networks (RNNs). The fourth paper is a review of an important branch – Graph NNs (GNNs). The last paper is concise and discusses recent, troubling trends in forecasting research. In the following, I concentrate on issues that I consider interesting and important.

First, I would like to note that generally, review papers follow a convention of using the “Deep Learning” phrase to describe the use of neural networks, whether they are truly deep or not (an unofficial definition of deep is having more than two hidden layers). It is actually incorrect, as time series forecasting, in contrast to image or language processing, typically operates on much smaller datasets and can often be done with shallow models, e.g., with two layers.

3.1. History

Let me quote from (Benidis, et al., 2022): “The history of NNs starts in 1957 (Rosenblatt, 1957) and in 1964 for NNs in forecasting (Hu & Root, 1964). Since then, interest in NNs has oscillated, with upsurges in attention attributable to breakthroughs. [...] Examples of such breakthroughs include (Rumelhart, Hinton, & Williams, 1986) that popularized the training of multilayer perceptrons (MLPs) using back-propagation, [...] convolutional NNs (CNNs) (LeCun & Bengio, 1995) and Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) cells [...]. Despite these advances, NNs remained hard to train and difficult to work with. Methods such as Support Vector Machines (SVMs) (Boser, Guyon, & Vapnik, 1992) and Random Forests (Ho, 1995) that were developed in the 1990s proved to be highly effective [...] and were supported by attractive theory.”

Thus, before the end of the 20th century, LSTMs and CNNs were already available. However, in the M3 Forecasting Competition, conducted in 2000, NN algorithms performed poorly (Makridakis & Hibon, 2000). I believe that the primary reason, probably even more important than the immaturity of NN tools, was the incorrect approach of trying to fit separate models for each time series. This approach was suitable for relatively simple statistical models, such as Exponential Smoothing, which have just a handful of parameters. However, it is the wrong approach for NN models with a much larger number of parameters – NNs work much better as global models, trained across many time series.

Another observation: yes, SVMs were attractive theoretically, but in practice, Tree-Based Algorithms (TBA), like Random Forest, were almost always beating them. Nonetheless, this theoretical attractiveness was a significant reason why TBAs were underutilized. It took years

of their dominance in Kaggle competitions before the scientific community reluctantly gave up on SVMs.

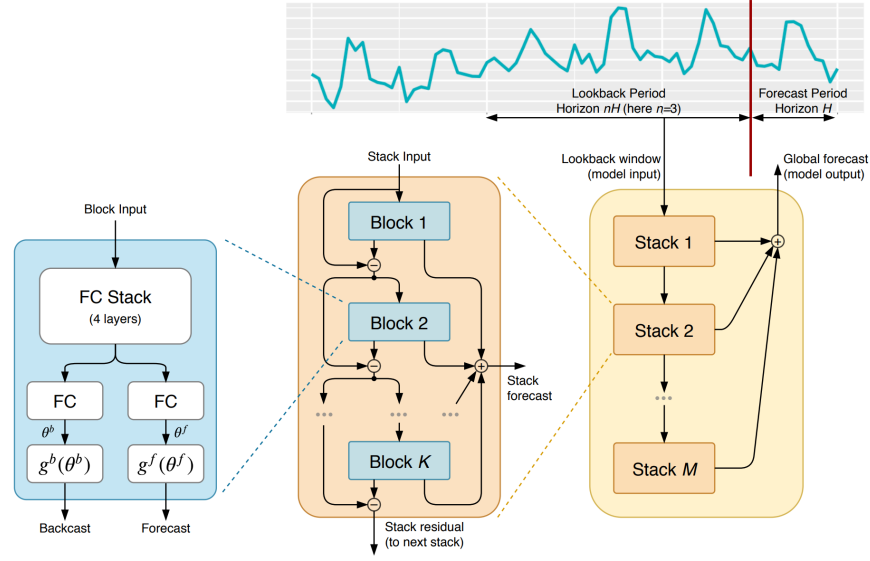
Back to NNs. The review (Benidis, et al., 2022) states that the M4 Forecasting Competition (Makridakis, Spiliotis, & Assimakopoulos, 2020), where my NN model won, was a pivotal development that convinced the skeptical scientific community of the use of NNs in forecasting. However, there is an interesting missing piece here. I won the IEEE Computational Intelligence in Forecasting Competition (Stepnicka & Burda, 2015) in 2015. To the best of my knowledge, it was the first time an ML/NN model won a forecasting competition against an expertly done ensemble of classical statistical models. However, the news did not reach the broader forecasting community until three years later (except for Prof. Bergmeir and Hyndman, who were involved in producing the classical algorithms benchmark for the competition). The review does not mention it. This shows the fragmentation of the scientific and engineering fields. I believe the next significant step was the emergence of deep architectures, which were simple and did not rely on earlier baroque training procedures (Hinton, Osindero, & Teh, 2006), but rather on the RELU activation function (Krizhevsky, Sutskever, & Hinton, 2012). This allowed the architecture to go deeper than one or two hidden layers. Three years later, the ResNet paper (He, Zhang, Ren, & Jian, 2015) allowed extending the depth to hundreds of layers. Although deep networks are not strictly necessary for time series forecasting, using ResNet-style shortcuts proved to be very useful when building even a few-layer deep networks with non-RELU activation functions, such as LSTM cells.

3.2. Architectures

3.2.1. MLPs

Multi-Layer perceptrons were introduced (Rumelhart, Hinton, & Williams, 1986), but they were not successful in time series forecasting. The primary reason, as I stated earlier, was the researchers' tendency to use separate models for each time series. Additionally, due to the use of squashing output functions like tanh and sigmoid, the depth was limited to a maximum of two hidden layers. Interestingly, recently, with some architectural improvements, including patching (splitting time-series derived inputs into semi-independently processing chunks, an idea seemingly taken from (Nie, Nguyen, Sinthong, & Kalagnanam, 2022), but actually not new, e.g., I used a version of it years ago) and inclusion of ResNet-style shortcuts and increase of the computational capabilities (with GPUs), the MLPs are back in vogue, e.g., Tiny Time Mixers (Ekambaram, et al., 2024).

A separate section is required for N-BEATS (Oreshkin, Carpo, Chapados, & Bengio, 2020). This model is popular and performs well. It can be viewed as an advanced MLP network with RELU nonlinearities. Its peculiarity is having two forecasting branches per block, one providing normal output, to be added to outputs of other blocks to form the final forecast, and another, which backcasts its inputs, to be subtracted from the block input, see the graph below:



The loss function is a weighted combination of the final forecast error and the average loss of the backtesting branch. Note that there are four FC layers (linear with RELU activation) that are shared, and only the last FC specializes the response into forecasting and backcasting.

The paper stated that their work enhanced my M4 results, which may have been true initially. However, I later created a new version of my M4 code that outperformed their reported findings. Unfortunately, I never got around to writing a paper about this improvement.

3.2.2. Recurrent Neural Networks

The first RNNs emerged relatively recently in the 1990s; they had a simple structure, with a state updated in every step based on its previous values and the current input, and the output was a squashed, linear-transformed state. For example, the Elman RNN (Elman, 1990) has the following recurrent formulas:

$$\begin{aligned} h_t &= \sigma(W_i \cdot h_{t-1} + V_i \cdot x_t + b_i) \\ z_t &= \tanh(W_o \cdot h_t + b_o) \end{aligned}$$

where z_t is the output at time t , W_i, W_o, V_i are matrices, and b_i, b_o are vector parameters to be learned.

The updates of parameters happen by unrolling the temporal computational graph, so for a larger number of steps, calculations become slower, and more importantly, potentially unstable: the gradient may explode, although more likely, due to the squashing functions like tanh and sigmoid, will vanish, so errors at early steps will hardly matter.

This problem was addressed with the publication of the LSTM architecture (Hochreiter & Schmidhuber, 1997). Here, the state is updated in a more sophisticated way:

$$\begin{aligned} i_t &= \sigma(W_i \cdot h_{t-1} + V_i \cdot x_t + b_i) \\ o_t &= \sigma(W_o \cdot h_{t-1} + V_o \cdot x_t + b_o) \end{aligned}$$

$$\begin{aligned}
f_t &= \sigma(W_f \cdot h_{t-1} + V_f \cdot x_t + b_f) \\
\tilde{C}_t &= \tanh(W_c \cdot h_{t-1} + V_c \cdot x_t + b_c) \\
C_t &= i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \\
h_t &= o_t \odot \tanh(C_t) \\
z_t &= h_t
\end{aligned}$$

Where i_t, o_t, f_t are, respectively, input, output, and forget gates – vectors of the state size with values between 0 and 1. Input and forget gates control the extent to which the old state is regenerated and the extent to which it is forgotten (reduced in size). The output gate shapes the output, which is taken to be equal to the h -state, a kind of short-term state; however, there is another, long-term state, C .

Almost 17 years later, a minor simplification was published called GRU:

$$\begin{aligned}
u_t &= \sigma(W_u \cdot h_{t-1} + V_u \cdot x_t + b_u) \\
r_t &= \sigma(W_r \cdot h_{t-1} + V_r \cdot x_t + b_r) \\
\tilde{h}_t &= \tanh(W_h \cdot (r_t \cdot h_{t-1}) + V_h \cdot x_t + b_h) \\
h_t &= u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1} \\
z_t &= h_t
\end{aligned}$$

We see the removal of input and forget gates. The former was replaced by a u (update) gate, and the latter by $1 - u$ value. Additionally, the output gate was removed, but replaced by the r -gate. In my experience, GRU usually performs worse in forecasting tasks. I mention it because I derived several new RNN cells by merging LSTM and GRU cells and adding my own ideas. It is detailed in the following chapters.

A modern RNN will contain one or more, often two or three, LSTM-style cells, followed by a linear layer. Generally, it works well if one follows some basic guidelines. For example, in contrast to the original application, text processing, where the step is always one word and the output is also one word, in time series forecasting, the step should be related to the problem specification. For instance, when forecasting the next 48 hours of energy consumption once a day, the step size should be 24; there is no reason to execute 24 steps if we do not care about the intermediary forecasts. Similarly, the entire output should usually be forecasted at once, forecasting a vector rather than a scalar.

In some circumstances, the encoder-decoder architecture (Sutskever, Vinyals, & Le, 2014) is used. Here, an RNN called an encoder steps through the series using some time step, and then, at the start of the forecast, a decoder, another RNN, takes over, executing one or more forecasting steps. The encoder passes its last output to the decoder. However, if the two have the same architecture (not weights, just architecture, such as the number of layers, size, etc.), the encoder can also pass its states to the decoder. Hence, the potential problem of an informational bottleneck (all information must be encoded in the last output of the encoder) becomes significantly less of an issue. The encoder-decoder architecture is particularly beneficial for long-term forecasts that exhibit strong seasonality and calendar events. For example, a 52-week forecast can be achieved with 13 decoder steps of 4 weeks, where the

decoder receives compact information (covering just one decoder step) about holidays, events, and other relevant details. This allows its responses to be sharper and more targeted.

Generally speaking, RNNs work well in time series forecasting; however, the attention of scientists has shifted to other architectures. This is likely because RNNs have limitations for LLM tasks.

3.2.3. Convolutional Neural Networks

This is of course a very successful family of models in image processing (LeCun & Bengio, 1995). However, there is no apparent reason why architecture designed to detect, say, an eye in any place of an image would work well for forecasting time series where the sequence and location are crucial. Nonetheless, for a few years, numerous papers were produced, and considerable effort was invested in achieving rather underwhelming results. For example, one of the most recent, most cited, and implemented in Nixtla library, CNN system called TimesNet (Wu, et al., 2022) lists its claimed to be great, results on the M4 Competition dataset, achieving an average OWA (the Competition metric) of 0.851, beating several benchmarks, including another Temporal Convolution Network and several Transformer-based systems. However, my winning solution, without the benefit of seeing actuals, achieved 0.821, and 0.851 would put TimesNet in seventh place, behind much simpler statistical methods (Bergmeir, 2024). On other long-term forecasting datasets, it barely outperforms the straightforward DLinear algorithm of (Zeng, Chen, Zhang, & Xu, 2023).

The fundamental problem is the base feature of CNN: its positional near-invariance. Consider that the standard 1-D convolution is a window of some size k , which is moved across the input window at each step, calculating the dot product and identifying a pattern encoded in the kernel's weights. However, it does not make sense to try to find the same pattern at the beginning and end of the input window. Even if the same pattern exists, its meaning is likely to be very different, depending on the position within the input window. A more sensible approach would be a set of non-overlapping or partially overlapping MLPs that are independent and cover the input window. However, this would not be a CNN, not even a dilated CNN.

3.2.4. Graph Neural Networks

This is a diverse family of models suitable for many situations where we expect that neighbors will influence, perhaps after some delay, the values recorded at a particular position/node, e.g., when forecasting street traffic or air pollution in a city. These models are beneficial for high-frequency forecasting, where the time grain of the forecast is similar to the time it takes for influence to propagate between nodes in the vicinity. In such cases, the forecast algorithm for each node can be simpler, as the complication is distributed.

If the time grain is larger, the information about the history and current values of some nodes can still be helpful. However, we can no longer rely solely on the physical layout and must consider all other nodes as potential “neighbors” – i.e., nodes that aid in forecasting a particular node.

Finding a neighborhood for each node automatically, as expressed in an adjacency A matrix, can be challenging. One approach is to utilize a distance or similarity measure between nodes, such as correlation or, better, transfer entropy. Another way is to create embeddings, either of node identity or its parameters (Corradini, et al., 2025), e.g., for the former case:

$$A = \text{softmax}(\text{ReLU}(E_1 E_2^\top))$$

where E_1, E_2 are matrices with node embeddings, randomly initialized.

Note that what would make E_1 and E_2 move is matrix A 's participation in some forecasting formulas and then the gradient descent updating all parameters (including E_1 and E_2), based on some forecasting error.

The adjacency matrix can be static during inference. However, some authors attempt to make it dynamic, which, in theory, is an attractive idea as it may reflect a changing situation, e.g., some roads may be blocked or made one-directional during the afternoon. However, it is not easy to achieve some plasticity without excessive variability (Corradini, et al., 2025).

The review paper (Corradini, et al., 2025) divides spatio-temporal GNNs into three categories: Recurrent, Convolutional, and Attentional. These categories reflect its handling of the ‘‘spatio’’ part – the temporal part; forecasting is done, as usual, with MLPs, CNNs, or RNNs.

The first category, recurrent, is the oldest and apparently less popular nowadays. It iteratively calculates state h of a node u with the following general formula:

$$h_u^{(k)} = f\left(x_u, \left\{x_v, r_{(u,v)}, h_v^{(k-1)} \mid v \in \mathcal{N}(u)\right\}\right)$$

where k is the iteration step, x_u is the vector of node u attributes, $r_{(u,v)}$ are the attributes of the connection (edge) between u and v , and $\mathcal{N}(u)$ is the neighborhood of node u . Function f has to be a contraction mapping.

The second category, convolutional, is an extension of the recurrent category. Here, there may be different parameters in each iterative step, seen also as a layer:

$$H^{(k)} = f_k\left(\tilde{\mathcal{A}} H^{(k-1)} \Theta_{k-1}\right), \quad \tilde{\mathcal{A}} = \hat{\mathcal{D}}^{-\frac{1}{2}} \hat{\mathcal{A}} \hat{\mathcal{D}}^{-\frac{1}{2}}, \quad \hat{\mathcal{A}} = \mathcal{A} + I, \\ \hat{\mathcal{D}}_{ii} = \sum_j \hat{\mathcal{A}}_{ij}, \quad H^{(0)} = X,$$

where $H^{(k)}$ is a k^{th} linear layer (matrix), $H^{(k)} \in R^{n*dk}$, n is the number of nodes (series), dk is the size of embeddings (model size), and $\Theta_{k-1} \in R^{(dk-1)*dk}$ is a trainable parameter of the k^{th} layer.

The last category is Attentional GNNs. Here, the recurrent formulas involve attention between a node and its neighbors. For example:

$$z_u^{(k)} = W_k h_u^{(k)} \\ \epsilon_{uv}^{(k)} = \text{LeakyReLU}\left(a^{(k)\top} \left(z_u^{(k)} | z_v^{(k)}\right)\right)$$

$$\alpha_{uv}^{(k)} = \frac{\exp(\epsilon_{uv}^{(k)})}{\sum_{w \in \mathcal{N}(u)} \exp(\epsilon_{uw}^{(k)})}$$

$$h_u^{(k+1)} = f\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{(k)} z_v^{(k)}\right)$$

where W_k is a matrix model parameter, $\parallel$ is a concatenation operator, and $a^{(k)\top}$ is a vector model parameter. The attention can take other forms, e.g., Transformer-style.

In summary, GNNs encompass a variety of models that are particularly effective for prediction tasks when the underlying problem can be represented as a graph. However, the algorithms tend to be slow, especially when an automated creation of the adjacency matrix is required, as this usually involves N^2 steps, where N is the number of time series. Moreover, if the A matrix needs to change over time, these calculations must be repeated frequently. In an unpublished paper (Sriramulu, Bergmeir, & Smyl), my colleagues and I demonstrate that an RNN with a Context mechanism can outperform GNNs while being more efficient.

3.2.5. Attention

Attention is a mechanism, not a comprehensive architecture, but it is discussed here as it is utilized in both RNNs and Transformers.

The first paper to propose a form of attention was by (Bahdanau, Cho, & Bengio, 2014), which focused on text processing. It is attention over RNN states. At each decoder step, an additional synthetic state is calculated as a weighted combination of the encoder states, h_j :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

where T_x is the number of encoder's steps, and α_{ij} are normalized weights:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

of alignment scores, between encoder state h_i and decoder state s_{i-1} ; $e_{ij} = a(s_{i-1}, h_j)$.

In the cited paper, they used the following function, which would become called additive attention:

$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$

where $v_a^\top$ is a vector, and W_a, U_a are matrix parameters to learn.

Generally speaking, this kind of attention between current and past states of an RNN is a good idea. Unfortunately, it is computationally costly. However, in multi-layer RNNs with

dilated cells, this mechanism can be approximated with a much smaller computational penalty. It is described in the coming chapters.

Attention is crucial in Transformers (Vaswani, et al., 2017). Transformers are multilayer NNs where each layer takes in several vectors (in the context of text processing, each vector is an embedded word) and outputs the same number of transformed vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, V are matrices resulting from multiplying the input matrix (all the input vectors stacked) by three different matrix parameters W_q, W_k, W_v .

In words: for every input vector, the output is a weighted combination of all value vectors (from V) with weights being the normalized dot product of its query vector (from Q) with all key vectors (from K).

3.2.6. Transformers

Transformers were developed as an alternative to RNNs in text processing. RNNs struggled to process long text fragments, required excessive memory, and, due to the unrolling of temporal steps, which creates a complicated computational graph, were unable to utilize Graphical Processing Units (GPUs) efficiently. Transformers met the requirements, and the era of Large Language Models (LLMs) commenced.

However, note that while RNNs were created with a view to processing sequences, Transformers were explicitly created for text processing. Their Attention layer is permutationally invariant – the input vectors can be shuffled, and each will be transformed in the same way. This makes sense for text processing, as, for example, “John loves Merry,” “John is in love with Merry,” or “Merry is the love of John” differ in structure, even using some different words, but express much the same meaning. However, a series that starts with a value of 100, declines gradually to 1, and a series that starts with 1 and grows gradually to 100 are dramatically different series that call for very different forecasts. Admittedly, the original Transformer NN (Vaswani, et al., 2017) utilized some positional information by adding an equally long vector to each input vector, which is the result of a trigonometric function applied to the position. However, this mechanism is relatively weak, as shown in (Zeng, Chen, Zhang, & Xu, 2023). Transformers used for time series forecasting are close to permutation-invariant. Actually, after removing the positional information, most Transformer models improved their accuracy. It is remarkable that, in this paper, the “embarrassingly simple baseline” of a linear autoregressive model, DLinear, achieved better accuracy than almost all the popular and cited Transformers.

The data size in forecasting is significantly smaller than that used for the leading LLMs, and time series data inherently contain randomness. Therefore, the models do not require and should not be on the same scale as LLMs. Even the slowest and memory-hungry category of NNs used in forecasting, RNNs, are up to the task, in practically all circumstances. So, the

proponents of Transformers created a niche for themselves: long-term forecasting. They, for example (Nie, Nguyen, Sinthong, & Kalagnanam, 2022), work with a few datasets, each comprising a small number of long series, and use a long horizon, e.g., 720. Such an extended horizon is impractical, as any recipient of the forecast will of course summarize it to make it worthwhile, by e.g., creating monthly means, maxes, perhaps a few quantiles, etc. (Bergmeir & Smyl, 2023). These summaries could be forecasted themselves, with much higher speed and better accuracy, by other architectures.

Edging towards comical, in several earlier papers on transformers in forecasting, one of the favorite datasets was the currency exchange rate. Yes, forecasting a financial time series with a 2-year horizon! As expected, a naïve forecast bested all the sophisticated *formers (Informer, Pyraformer, Autoformer, etc.) (Bergmeir, 2024b). Equally *brave* is forecasting weather up to 720 hours ahead – another popular dataset.

Generally, it is a concerning trend – people are publishing papers reporting how their latest version of a particular architecture outperforms a previous version, but often omit basic benchmarks against naïve, established statistical methods, as well as other architectures. Moreover, they do it on datasets that make no practical sense.

However, I would like to mention two papers here. The first study (Nie, Nguyen, Sinthong, & Kalagnanam, 2022) proposed a practical approach that can be applied to other architectures and datasets, namely patching, which involves dividing the time series history into several chunks. They use these chunks to create vectors (tokens) needed by a Transformer-style NN. However, one can process them semi-independently with other architectures, including RNNs (our upcoming paper) and MLPs (Ekambaram, et al., 2024). It makes sense particularly if the chunk covers smaller seasonality, e.g., 7 days.

Another paper is on a rare case of a Transformer system that performs decently (even not very efficiently) – Temporal Fusion Transformer (Lim, Arik, Loeff, & Pfister, 2021). In this context, Transformer self-attention is applied to the states of an LSTM, resulting in a hybrid architecture. The strong performance observed can likely be primarily attributed to the capabilities of the LSTM.

3.2.7. State Space Models (SSMs)

They can be seen as simplified, typically linear, but potentially much faster versions of RNNs. They were billed as an alternative to Transformers for handling very long sequences in text processing (Gu, Goel, & Ré, 2022), as standard Transformers' computational costs grow quadratically with the sequence length. Note that SSMs were trying to be more efficient, operate on even longer sequences than Transformers can. However, this has very little to do with time series forecasting. As I argued above, super long-term forecasting tasks do not make sense.

Mamba (Gu & Dao, 2024), added a gating mechanism, as the previous formulation proved to be too simple, not capable enough. That made the SSM more complicated, but some wizardry in low-level coding allowed it to maintain good speed.

However, (Merrill, Petty, & Sabharwal, 2024) questioned the very existence of the state in these fast, linear SSMS. This makes sense to me – if one can parallelize a recurrent computation, it suggests that the state is not really needed.

As in the case of Transformers, despite a fundamental mismatch between attributes of SSMS and what is needed in time series forecasting, several forecasting papers attempted to use them, with the same, rather useless, datasets. The success was moderate, with Mamba-based models performing on par or slightly better than Transformer-based models, see e.g. (Wang, et al., 2025).

4. Paper [D1]: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting

4.1. Overview

This invited paper (Smyl, 2020) describes my winning entry in the M4 (fourth edition) of the Makridakis Forecasting Competitions – the most important and very influential sequence of competitions in the forecasting community (Hyndman, 2018). The M3 Competition, held in 2000, confirmed the findings of earlier competitions that simple statistical models, such as the Theta method, perform equally well or better than more advanced models, like ARIMA. It also added a new finding that ML models typically performed worse. My system was a surprise that went against several expectations (Makridakis, Spiliotis, & Assimakopoulos, 2020) including that ensemble of statistical methods will win the competition, simple statistical benchmarks will not be far behind a winning entry (actually, a point forecast metric was 8.4% better) and that prediction intervals will be underestimated considerably (actually, the PIs coverage was close to perfection).

4.2. Dataset

The dataset contained 100,000 series across several granularity subsets: hourly (414), daily (4227), weekly (359), monthly (48,000), quarterly (24,000), yearly (23,000) (Makridakis, Spiliotis, & Assimakopoulos, 2020), of unknown origin, without any regressors, not even dates and times. There were two tracks: one traditional point forecasting to minimize a metric that was a weighted combination of sMAPE and MASE, and a prediction intervals track to minimize Mean Scaled Interval Score (MSIS) (Gneiting & Raftery, 2007). I built separate models for each track and grain, resulting in 12 NN models.

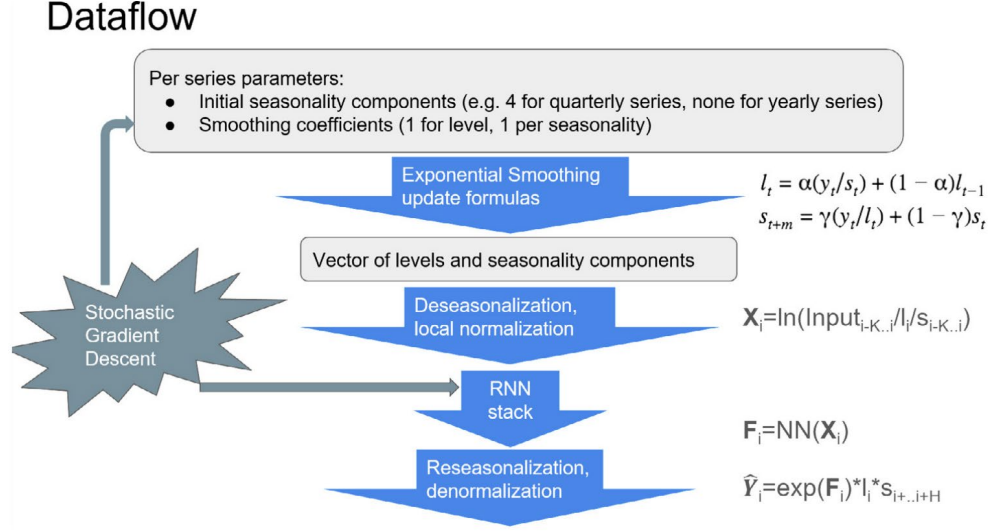
4.3. Model architecture

As mentioned above, there were many models; however, in each case, it was an RNN with some on-the-fly input pre- and post-processing, utilizing simplified Exponential Smoothing formulas. The diagram below provides a high-level view of a system with single-seasonality (quarterly and monthly data).

The NN architecture varied between subsets (granularity), with experimentation playing the most important role in choosing between them (see Fig. 3 in [D1]). Generally speaking, it can be described as a multi-layer RNN with LSTM cells and residual connections.

Architecture used for quarterly data was proposed in (Chang, et al., 2017) and although in the Competition I also used a modified LSTM cells proposed by (Kim, El-Khamy, & Lee, 2017) (for monthly series) and LSTM cells with attention on past states for yearly series, it works well in many contexts, and it stood the test of time. The choice of dilations follows some rules of thumb: the first layer should focus on the most frequently changing component, so its dilation equals 1 (it is a standard LSTM cell); a geometric progression of dilation steps is often a good idea; and at least one of the dilations should be equal to the seasonality of the series.

With several layers of increasing dilations, one can argue that this approximates a much more computationally intensive architecture that uses attention over past states.



4.4. Main innovations

The forecasting system employed several new or not widely adopted mechanisms. Looking back, the most important ones that have stood the test of time are as follows:

- 1) **Global model.** Earlier, largely unsuccessful attempts to apply ML methods to time series forecasting followed the established statistical methods' approach of fitting a separate model for each time series. This was not a good idea. ML models tend to have several orders of magnitude more parameters and benefit from learning across many time series, typically the entire dataset or a significant portion of it (a cluster).
- 2) **Adaptive normalization.** When global learning is used, and especially when a dataset is not huge, but contains a series of widely varying values, or the algorithm has a limited input range, as in the case of neural networks with squashing functions, such as LSTMs or MLPs with tanh or sigmoid activation functions, it is necessary to normalize the data. Normalization can be categorized as:
 - a. global across all series,
 - b. global per series,
 - c. adaptive: per series per each time step.

The first type of normalization can only work if all series have a similar range, also across time. For example, when forecasting CPU utilization of a stable system, it can be scaled from a theoretical range of 0-100 to a 0-1 range. The second type may work if series are of different sizes, but relatively stable, not trending. Otherwise, a series growing very fast, say from 5 to 500, and another one growing slowly, say from 100 to 200 during the same period, may, after min-max normalization, appear very similar, and we lose some important information. Additionally, in both cases, a normalization whose

parameters are established in the training-only part, as it should be done to prevent leaking information, when applied to the validation or test area, may result in values being outside those seen during training, e.g., outside the (0,1) range (this kind of leakage is standard in many recent papers).

The final, adaptive normalization is the most general and was used by my models in the competition. In it, input and output values are normalized differently for each series and each time step.

- 3) **Seasonality modelling.** The M4 dataset contained several subsets, most of which were seasonal, e.g., monthly and quarterly. Explicitly modeling seasonality in ML systems is seldom done, as it is not straightforward, and people often assume that the ML algorithm can learn it automatically with sufficient training data and the provision of date- or time-derived features. However, the explicit modelling seldom hurts, and I believe that it was needed in the context of the M4 competition, where the series were drawn from many completely unrelated sources, e.g., some represented river overflows, some ATM withdrawals, others demographics, etc., so there were no common seasonality patterns among them.

The modelling followed somewhat simplified Holt-Winters method of Exponential Smoothing family (Hyndman, Koehler, Ord, & Snyder, 2008) with the following formulas:

$$l_t = \alpha(y_t/s_t) + (1 - \alpha)l_{t-1}$$

$$s_{t+m} = \gamma(y_t/l_t) + (1 - \gamma)s_t$$

where l is a state variable called *level* (a smoothed version of y), s are multiplicative seasonality coefficients (so they tend to stay around 1), α, γ are smoothing parameters between 0 and 1, and t is the time step. Only α and γ are parameters that need to be fitted.

Although we also need to arrive at the initial values of the seasonality coefficients. Statisticians have several ways to do it; here, we treated the initial values, e.g., $s_1, \dots, s_{12}$ for the monthly series, as per-series parameters, initialized at ones, but fitted by gradient descent during the training.

Knowing the initial values of s , the smoothing parameters, and actuals allows us to calculate the levels and seasonality coefficients. What was innovative and gave the method its “ES-RNN” name was that the per-series parameters α and γ were fitted using the exact gradient descent mechanism that updated the NN parameters. There was no separate preprocessing step run before the main program. For the hourly series a more complicated model with two parallel seasonality modeling mechanisms (24 hours per day and 7 days per week) was used; however, in retrospect it was not a good idea – it would have been better to use a single 24·7 seasonality model. (Provided, however, that the series had 168 hours per week seasonality, which turned out not to be true for all series; some of them had shorter seasonality, likely due to missing weekends.)

For non-seasonal subsets, e.g., yearly, a simplified Holt linear method (Hyndman, Koehler, Ord, & Snyder, 2008) was used:

$$l_t = \alpha y_t + (1 - \alpha)l_{t-1}$$

and the level l_t was used, as in all versions, for adaptive normalization (see below).

- 4) **Global and per-series parameters.** My system was primarily a global one, but it also used a small set of per-time series parameters (smoothing parameters). This type of modeling is only possible in dynamic computational graph systems, where the graph is distinct for each batch. It includes, apart from global parameters such as the main RNN and some embedding layers, also the per-series parameters, but only for the series that are part of this batch. At the time of the M4 Competition, only the Dynet system (Neubig, et al., 2017) had this capability.

Per-series parameters work very well when the number of series is relatively small, as each series needs to be part of a training batch at least several times. For large datasets, it is less likely to work.

- 5) **On-the-fly preprocessing and postprocessing.** As mentioned above, there was no separate preprocessing step before the main run. Instead, preprocessing happened on-the-fly during training and validation. Series were sampled randomly, and then – starting from its beginning or, for a long series, a first point after a truncation of the oldest values – adjacent input and output windows were moved one step at a time (see Fig. 1 in [D1]).

At step t , values of input and output were normalized by dividing by l_t in non-seasonal cases, and by $l_t s_i$ in seasonal cases. The index i reflects the position within the input or output window. So, the seasonality was mainly removed, and the NN could “concentrate” on the trend.

Finally, a $\log()$ transformation was applied to these preprocessed values. This “squashing” is often desirable, as it makes the model robust to outliers – whether real or caused by data collection problems. So finally, in the seasonal case, the input was:

$$x_i = \log\left(\frac{y_i}{l_t s_i}\right)$$

In the validation stage, the output of the RNN was “unwound” to create the forecast:

$$\hat{y}_{t+1..t+h} = \exp(RNN(x_t)) \cdot l_t \cdot s_{t+1..t+h}$$

in case of seasonal series, and:

$$\hat{y}_{t+1..t+h} = \exp(RNN(x_t)) \cdot l_t$$

otherwise, where x_t denotes the preprocessed input.

Please note that the whole horizon number of points was forecasted at once.

- 6) **Bespoke loss function.** The loss functions I used closely matched the competition metrics. For example, in the case of the prediction interval task, it was the Mean Scaled Interval Score (MSIS) defined as (Makridakis, Spiliotis, & Assimakopoulos, 2020):

$$MSIS = \frac{1}{h} \cdot \frac{\sum_{t=n+1}^{n+h} (U_t - L_t) + \frac{2}{\alpha} (L_t - Y_t) 1\{Y_t < L_t\} + \frac{2}{\alpha} (Y_t - U_t) 1\{Y_t > U_t\}}{\frac{1}{n-m} \sum_{t=m+1}^n |Y_t - Y_{t-m}|}$$

where U_t is the predicted upper quantile (0.975), L_t is the predicted lower quantile (0.025), Y_t is the actual, and α is 0.05. The denominator is the average error of a naïve forecast over the in-sample period.

Instead of using the standard Pinball loss, as it is usually done for quantile regression, I used the formula above, although just the numerator, because my input and output values were already normalized.

Generally speaking, it is a good idea to create a custom loss function reflecting the forecasting goals, whether business or scientific.

- 7) **Level wiggleness penalty.** This is an additional penalty added to the primary loss to increase the smoothness of the on-the-fly deseasonalized time series input. It is calculated as follows. First, we conduct the on-the-fly preprocessing as described in p. 5 above. The output includes the vector of levels l . We try to reduce the wiggleness of this vector as follows:

1. Calculate the logarithms of ratios 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/batch,
4. Add, with some weight, to the loss function.

Reducing the level vector variability enables a better fitting of the initial seasonality levels and smoothing coefficients. Therefore, after normalizing the input values by l_t , we expect the input to become a smooth, trending curve, and the NN does not need to deal with seasonality. This was important in the M4 Competition, as the series originated from diverse and unknown sources, with no expectation of having any common seasonality. However, in subsequent forecasting projects, I found it to be useless, and this idea was subsequently abandoned.

- 8) **Ensembling.** The system used several types of ensembling:

1. Simple – initialization and data subset. Six to nine identical programs were run, with NNs that were randomly initialized and trained on different data: randomly selected 50%. In my later work, when working with large datasets, I would tend to use sampling, so each training epoch of each program would see a different sequence of series (randomly sampled). This approach is more flexible than the upfront data split. Nonetheless, it performed well. The final forecast would be a median of all, say, nine forecasts.

2. Stage of training. Averaging forecasts produced during the last few epochs. It worked in the M4 competition, but not much afterwards, so I will not dwell on it; however, it is discussed more in one of the subsequent chapters.
3. Ensemble of Specialists. It was computationally intensive, so it was used only on smaller datasets. The main idea was that, if it were possible to split a dataset into several similar subsets and then train a specialist model on each of these subsets, the overall error would likely be smaller.

Of course, how to make the splits is not apparent. In the case of large datasets, such as customer behavior across various continents, one can follow what makes business sense and is intuitive, e.g., splitting the data by geography or customer income. However, this was impossible in the M4 competition – there was not much metadata provided, and the membership in the provided groups was very skewed.

Another approach is to split the data based on features derived from the series, such as volatility and strength of seasonality (Bandara, Bergmeir, & Smyl, 2020). However, it is unclear which grouping would be the best for a particular dataset and forecasting algorithm.

The Ensemble of Specialists was proposed earlier (Smyl, 2017). The algorithm is summarized in Section 3.2.2 of [D1]. The final forecast for a particular series is the average of the forecasts produced by the top N models. The central assumption here is continuity: if a particular model is good at forecasting the in-sample part of the series, it will likely display accurate results for the out-of-sample part as well – see the diagram shown in Fig. 4 of [D1].

This algorithm performed well in the Competition, improving accuracy by up to 3%. However, subsequent experimentation on other datasets showed that its performance is uneven, often failing to provide any improvement. Nonetheless, it was a precursor to better algorithms of this type, which have yet to be published.

4.5. Results

The competition had two tracks: point forecast and prediction intervals. I won both. The fact that an ML-based method outperformed an expertly crafted ensemble of statistical methods was unexpected (Makridakis, Spiliotis, & Assimakopoulos, 2020). Even more surprising was the margin of victory: in the point forecast track, my results outperformed a benchmark of statistical models by 8.6% and exceeded the second-placed team, who used a hybrid ML-statistical approach, by nearly 2%.

In the Prediction Intervals track (predicting the 2.5th and 97.5th percentiles), my results were even stronger, improving by 49% over the benchmark and nearly 9% over the second-place solution. The prediction interval coverage, unlike typical results of statistical algorithms, was not too narrow but close to perfect, at 94.78%. The organizer, Prof. Makridakis, stated that the “M4 Competition ended winter in the forecasting”. It is not an exaggeration to say that my winning pioneered widespread usage of NNs in forecasting.

5. Paper [D2]: Machine learning methods for GEFCom2017 probabilistic load forecasting

5.1. Overview

This was one of my earliest papers that utilized Machine Learning, and neural networks in particular, for time series forecasting. It described a system that was ranked second in the first stage of the Global Energy Forecasting Competition 2017 (Hong, Xie, & Black, 2019). I participated as a member of a two-person team; the final model was an ensemble of a modern Multilayer Perceptron (MLP) neural network and a few tree-based algorithms. This paper chronologically preceded the M4 paper described in the previous chapter; it is presented here because it applies some of the concepts elaborated there.

5.2. Main innovations

When I wrote this paper, I believed that the “Escape Seasonality Trick,” as described below, was a novel concept. However, a more diligent search in the literature might have revealed its prior use.

5.3. Data and task

There were 10 time series of hourly electrical loads of regions served by a company called ISO in New England, USA. The historical data spanned nearly 15 years. The competition was a rolling one, and the task was to predict a monthly load starting 2 months later, covering 5 months in spring. The gap made it impossible to use recent weather information (although we did find that some long-term temperature patterns were useful).

5.4. Data preprocessing

As the Competition’s prediction task was limited to the spring months, we decided to limit learning to this output period. This, in turn, led us not to use RNNs, which have memory (state) and would not tolerate sudden gaps in the time steps.

Each model we used was a global one, trained on all series. The output window spanned 30 days, each with 24 hourly values, totaling 712 points. We considered it too long, even for a well-crafted neural network. Additionally, the electricity load hourly data exhibit complicated seasonality, with yearly (due to changing seasons and temperatures), weekly (weekdays differ from weekends), and hourly cycles. The last two periodicities are intermingled and cannot be modelled separately; therefore, it is better to view them as a single $24 \cdot 7 = 168$ -hour seasonality. Learning such a seasonality would be challenging, so instead, we deployed what I call the escape seasonality trick: creating input-output pairs that include only points of the same day of the week and hour – see the graph in Fig. 2 in [D2].

This allows the learning algorithm to focus on longer trends (e.g., season-related) and avoid being burdened by intra-week fluctuations. This is a good trick, but its drawback is that it

multiplies the number of forecasting steps: a single forecast is actually assembled out of 168 independent forecasts. It was acceptable in the Competition as there were only 10 series involved, but this could be a problem with much larger datasets.

To cover 30 days, we forecasted five 168-hour points. The competition required forecasting nine percentiles (10, 20, ..., 90), and due to the limitations of the tool used at the time (Microsoft's CNTK – deep learning toolkit that describes neural networks as a series of computational steps via a directed graph), separate runs were required for each percentile.

As the series were relatively stable, without any notable long-term trend, we utilized a global, per-series normalization.

5.5. Neural network and ensemble

The system was an ensemble of three tree-based and one neural network-based forecasters. One of the tree-based systems (created by my colleague) had its own preprocessing, while my other two tree-based systems and the NN shared my preprocessing (see Fig. 1 in [D2]). The final forecast was a weighted average with weights inversely proportional to validation errors.

We did not use an RNN in this project; instead, we used an MLP, but a modern one with ResNet-style shortcuts (He, Zhang, Ren, & Jian, 2015). This allowed it to have more than one hidden layer with a normal squashing function (tanh), see Fig. 4 in [D2]. A general remark: The ReLU activation function is not great for the regression needed for time series forecasting. If tanh() cannot be used for some reason, LeakyReLU is a better alternative.

From my current perspective, the NN was not properly optimized: we did not check and fine-tune the coverage (empirical percentage of times where an actual is below a predicted quantile), and so did not use *training quantile levels*, which are a bit distorted (shifted) versions of the desired quantile levels, shifted in a way to achieve the correct coverage. As a result, the NN tended to under-forecast. Luckily, the other methods had the opposite effect, so the average was decently calibrated and performed well. Microsoft Fast Forest (<https://msdn.microsoft.com/en-us/library/azure/dn913093.aspx>), a version of Quantile Random Forest (QRF) (Meinshausen, 2006), performed best; however, the NN was not correctly tuned.

6. Paper [D3]: A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting

6.1. Overview

This paper describes an attempt to apply my M4 Competition monthly model to the problem of midterm electric load forecasting. In both cases, the data grain is month, and the horizon is also similar: 18 months in M4 and 12 months here. However, the M4 dataset comprised 48,000 series, whereas the midterm load dataset contained only 35 series (from European countries), spanning lengths of between 5 and 24 years. While accuracy was good, ablation studies revealed that some of the techniques that performed well on the much larger M4 dataset were ineffective on this dataset, which is two orders of magnitude smaller and more homogeneous.

6.2. Model

It was close to my M4 Competition monthly point forecast model: four dilated LSTM-style layers with residual shortcuts starting from the second layer, followed by a linear layer on top, with some Exponential Smoothing formulas embedded. Fig. 1 in [D3] presents a block diagram of the model, and Fig. 6 shows the NN architecture (omitting, however, the residual shortcuts).

Instead of the modified LSTM cells proposed by (Kim, El-Khamy, & Lee, 2017), which were used in my monthly M4 model, a simpler dilated LSTM NN model by (Chang, et al., 2017) was employed, based on insights gained after the M4 Competition.

The same embedded Exponential Smoothing formulas for a single seasonality were used, see Section 4. Since the number of series was small, it was easy to fit per-series parameters (smoothing coefficients and 12 smoothing factors).

The ensembling was similar to one of the M4 models: a stage of training (the last few epochs), a data subset (say, three sets, each composed of 2/3 of all series), and a few models differing in their random initialization.

6.3. Results

The model's accuracy was better than that of most comparative models, such as ARIMA, ETS, MLP, and the standard LSTM network, but worse than that of some ML and ETS hybrids, e.g., the Fuzzy Neighborhood-ETS model. However, the differences were not significant, as seen in Table II in [D3], where my model is listed last, labeled as ETS+RD-LSTM.

6.4. Ablation Studies

For me, it was the most interesting part of the paper, learning which parts of the system were included (after a lot of testing) in the M4 system was not helpful here.

The stage of training ensembling (aggregating forecasts done by the last few epochs) was not important. It was somewhat expected; in my experience, it seldom works.

The data subset ensembling (building separate models on, e.g., three datasets composed of 2/3 of all series) also did not yield satisfactory results. I think it may only work when the dataset is of medium size (like in the M4 Competition), because for small datasets, each series is precious. We lose too much information by not using all of them, and for massive datasets, a better strategy is to sample a number of series, say 10,000, and process them one “epoch”.

The error increased when all ensembling was removed, as expected, but the increase was slight; the Wilcoxon test indicated no significant difference. It was surprising – perhaps the dataset was just too small?

Removal of the residual connections around cells increased error by around 8% and was statistically significant, as expected. We used four layers; there are just too many squashing functions in an LSTM-style cell. We are getting a vanishing gradient problem without the residual connections.

In the M4 monthly model, the dilations were 1, 3, 6, and 12. A rule of thumb is that they should be in (perhaps geometrically) increasing order, and one of the dilations should be equal to the central seasonality of the series, which is 12 in this case. It turned out that actually 1, 3, 6, 9 dilations worked better. The cause probably lies again in the small size of the dataset – the 4-layer model is unnecessarily large in this case. Note that two- or even single-layer models are not significantly worse – see Table III in [D3].

The loss function was a weighted combination of standard pinball loss and wiggleness loss – see (19) in [D3]. The second term proved to be not useful; actually, it increased the forecasting errors a bit. As mentioned earlier, the wiggleness loss may be helpful only in certain exceptional circumstances, such as a medium-sized dataset composed of largely unrelated series.

The on-the-fly processing component that used the simplified single-seasonality formulas (see (1) in [D3]) worked well here; without it, the MAPE error increased over 5%. Although the series are related, the dataset is so small that the fitting of per-series parameters works well.

6.5. Conclusions

I think the main conclusion of the paper was that with a much smaller dataset, the model and ensembling should be simpler, but residual connections are always needed to avoid vanishing gradients. If the dataset is not huge so that per-series parameters can be fitted, it is worth trying them.

7. Paper [D4]: 3ETS+RD-LSTM: A new hybrid model for electrical energy consumption forecasting

7.1. Overview

The paper was a follow-up to the paper described in the previous chapter. It operated on the same dataset and used the same NN architecture, but with more elaborate pre- and post-processing, which substantially improved the results.

7.2. Main innovations

A more elaborate hybrid pre- and post-processing.

7.3. Main idea

The monthly electricity consumption at the country level does not exhibit strong trends. This allowed the following normalization (upper-left box in Fig. 1 in [D4]): for each time series, taking yearly chunks (12 points) and moving in yearly steps, calculating a mean value $\bar{z}$ and a measure of dispersion $\sigma = \sqrt{\sum_{j=1}^{12} (z_j - \bar{z})^2}$. Then, for every year indexed by i , the input values are normalized as follows:

$$y_{i,j} = \frac{z_{i,j} - \bar{z}_i}{\sigma_i} + 1$$

Fig. 2 in [D4] shows an example. This first preprocessing step is done offline. The resulting data (subgraph (d)) is fed into an online simple, Exponential Smoothing-style deseasonalization (lower left box on the system architecture) using following update formula:

$$s_{t+12} = \beta y_t + (1 + \beta)s_t$$

The first 12 seasonal components ($s_1 \dots s_{12}$) and β are per-series parameters that are fitted by the gradient descent. Having calculated the seasonal components (subgraph (e)), we now deseasonalize already normalized values:

$$x_t = \log \left(\frac{y_t}{s_t} \right)$$

arriving at normalized and deseasonalized series – example shown on subgraph (f).

Then, using windows size 12, we, with a monthly step, engage the RD-LSTM RNN, and then partially “unwind” and reseasonalize the output:

$$\hat{y}_t = s_t \exp(\hat{x}_t)$$

However, to fully “unwind” it, we also need to reverse the initial normalization, so the following formula is applied:

$$\hat{z}_{i,j} = (\hat{y}_{i,j} - 1)\hat{\sigma}_i + \hat{\bar{z}}_i$$

Note that because the input is, at least to some extent, detrended, the expected value and σ need to be taken from the forecasted period, not the past (input). A standard time series algorithm provides the forecasted values, Exponential Smoothing (top two red boxes in the architecture diagram).

7.4. The neural network

The neural network (RD-LSTM), its assembling, hyperparameters, etc., are very similar to those in the previously discussed paper, although without the wiggleness penalty, as it was not helpful for the mid-term electricity dataset.

7.5. Results

Table 2 in [D4] compares various methods, including classical time series algorithms such as ARIMA, and machine learning algorithms like nearest neighbors, GRNN, and MLP. The last two rows list metrics for the previous ETS+RD-LSTM algorithm and this paper's algorithm, 3ETS+RD-LSTM. The latter is clearly the best for all metrics, including median APE (Average Percentage Error), MAPE, IQR (Inter-Quantile Range of APE), and RMSE.

7.6. Conclusion

More advanced pre- and post-processing brought significantly improved results, despite the main NN algorithm remaining essentially the same. I believe the more thorough normalization (not just dividing by the *level*, but also normalizing the dispersion) was very useful in this case of an extremely small dataset, as the NN “saw” many more similar inputs.

8. Paper [D5]: ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting

8.1. Overview

This paper proposes an RNN-based forecasting system applied to short-term electricity load forecasting. In comparison to previous papers dealing with midterm electricity demand forecasting, this paper has been substantially modernized. Many of the tricks and improvements described there are, I believe, still state-of-the-art. And that's why I am describing it in more detail.

8.2. Main innovations

1. New LSTM-style cell called dRNNCell
2. Training quantiles
3. Random starts, warmup
4. Redefinition of epoch through a sublinear function of updates
5. Some per-series parameters (smoothing coefficients in the exponential smoothing formulas) adjusted dynamically, during time steps

8.3. Data and task

The dataset comprised several years' worth of hourly electricity demand data from 35 European countries (www.entsoe.eu/data/powerstats). We forecasted the expected value and 90% prediction interval, as well as the 5th and 95th percentiles, each composed of 24 hourly values ahead, once a day. There were some missing periods for specific countries, but generally speaking, we used 2016 and 2017 as the training periods and 2018 as the testing period.

8.4. Challenges

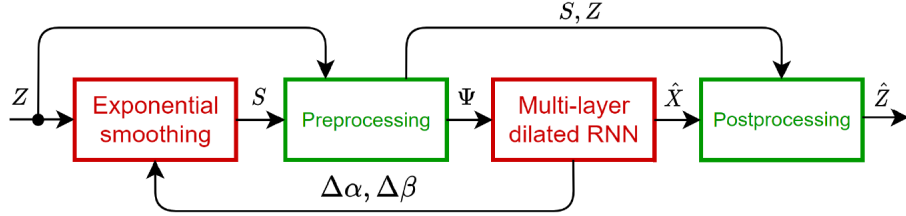
The electricity demand exhibits several seasonalities: yearly (due to changing seasons), daily (related to the days of the week), and hourly. Actually, the latter two seasonalities cannot be separated; for example, 5 PM on Wednesday has different patterns than 5 PM on Sunday, so they should be seen as a single 168-long seasonality. Additionally, temperature plays a crucial role, but we lacked a history of forecasts to include it in our modelling.

Time series are long (data is hourly), but there is a small number of them, 35. Each of the series has its own characteristics, different seasonality patterns and trend. The graph shown in Fig. 3 in [D5] illustrates daily, weekly, and yearly variations in the demand.

8.5. Model

Overview

Below is the block diagram of the system (single worker).



The time series data Z is on-the-fly preprocessed by the Exponential Smoothing component. Its smoothing parameters, α and β , and initial seasonality coefficients are per-series parameters; however, they are also dynamically adjusted by the RNN (a novelty in this work). The RNN operates on deseasonalized and normalized series values, along with additional regressors primarily derived from the dates. The postprocessing component “unwinds” the output to provide the final forecast of a single worker. However, note that we use an ensemble of several workers, each running the same code but with different initializations and “seeing” different, randomly selected sequences of training data – this is not shown in the diagram above.

Exponential smoothing component

The exponential smoothing component is similar to the one in [D1], see (1) in [D5], but the period is set to 168, so it represents a single weekly seasonality. The levels l , in contrast to the M4 models, are not used for normalization; their only role is to participate in calculations of the seasonality coefficients s . We use the average of time series values in the input window as the normalizer – it turned out to work better this way. The input window is quite large, 168 hours, so the average is stable.

The smoothing coefficients α and β are per time series parameters. We can afford to do this as there is a small number of series here, so each of them will be involved with many updates. For example, assume that we define the epoch as 2000 updates and the batch size is 4; then, each per-series parameter will be updated approximately $2000/35 \approx 57$ times per epoch.

Additionally, the smoothing parameters are getting adjusted by $\Delta\alpha_t$ and $\Delta\beta_t$, which are some of the RNN outputs:

$$\begin{aligned}\alpha_{t+1} &= \sigma(l\alpha + \Delta\alpha_t) \\ \beta_{t+1} &= \sigma(l\beta + \Delta\beta_t)\end{aligned}$$

Where $l\alpha$ and $l\beta$ are the per-series parameters (initialized to average fitted values observed in early runs, $l\alpha_0 = -3.5$, $l\beta_0 = 0.3$). So, the smoothing coefficients differ not only across series but also across time steps.

Preprocessing and postprocessing

It is generally similar to the one used in monthly or quarterly M4 models (see (3) in [D5]). However, the input is richer, including the current seasonality factors and several data-related features, such as day of the week (see (6) in [D5]). We did not have country-specific holidays available; however, the date-derived features should capture stable holidays, such as Christmas.

It is also worth noting that we are stepping by 24 hours, forecasting once a day, so the daily seasonality is easier to manage, as seen in Fig. 6 in [D5].

RNN

A significant innovation described in this paper was a new dilated RNN cell we called dRNNCell. The cell is derived from LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Cho, et al., 2014) cells, merged with the dilation idea of (Chang, et al., 2017), and it almost always improves the forecasting accuracy. It is designed to operate as part of a multilayer dilated RNN, and as in (Ben-Ari & Schwartz-Ziv, 2017), its output is split into “real output” y_t , which goes to the next layer, and a controlling output h_t , which is an input to the gating mechanism in the following time steps.

The cell uses two states, c -state (also called a cell state), which is close to the standard LSTM state, and h -state, which is the controlling state (also called a hidden state). At each time step t , the whole dRNNCell input is a concatenation of x_t , h_{t-1} and h_{t-d} , where x_t is a standard input at time t (either from a previous layer or an input to the RNN), h_{t-1} is the most recent h -state, and h_{t-d} is the delayed state ($d > 1$). Both c - and h -states are saved in a list to be used as delayed states. The size of the c -state is equal to the summed sizes of h -state and y -output, see chapter IIIC in [D5].

The RNN is composed of two such cells, with 2 and 7 dilations, respectively, and resnet-style shortcuts (He, Zhang, Ren, & Jian, 2015), followed by a linear layer, as shown in Fig. 8 of [D5]. A separate linear layer embeds date-derived features. The output consists of the central forecast (expected value), lower prediction interval, upper prediction interval, and the smoothing coefficients’ adjustments.

Loss function and training quantiles

We forecasted the expected value and two prediction intervals (5th and 95th percentiles). For each, we used the standard pinball loss function. However, the quantile levels q used in training were close to, but not identical to, the “official” or testing quantiles. Regarding the center or expected value, if we use $q = 0.5$, we are fitting the median, not the expected value. The distribution is likely non-symmetric and has a fatter tail. Therefore, if we want to use the pinball loss and aim to achieve the expected value, we need a slightly higher quantile, say 0.54, to obtain a prediction with an average bias of zero, which is the hallmark of the expected value.

Similarly, in the case of the other two quantile levels intended to produce the prediction intervals of 0.05 and 0.95, the training quantiles may differ slightly from the validation or testing values of 0.05 and 0.95. This can occur for several reasons, including the fact that the

training operates in log domain and the final results are aggregates (usually treated with median) of all workers' results. We adjust them based on the coverage of validation results, e.g., expecting that in 95% of cases, across all predicted series, at all time steps and horizons, the actual should be around 95% of the time below the predicted quantile. This part was not well-explained in the paper.

Thus, we have three losses, a final loss is a weighted sum, see (18) in [D5]

8.6. Training details

Random starts

Often, especially when the number of series is small, overtraining is the major problem while building NN-based forecasting systems. Here, we delay the onset of it by sampling, for each batch, the beginning of stepping through time. After a specified number of training steps (a hyperparameter, set to 50), we stop, even if we have not reached the end of the series.

Warmup steps

We used warmup steps – a number of initial steps over time, during which the forward step is executed but the loss is not calculated. It is important when using dilated RNNs, as it allows the c-state to stabilize and the h-state to be available for longer-dilated cells. The number of warmup steps I use is typically 2 or 3 times larger than the most significant dilation. Additionally, using exponential smoothing with seasonality provides an extra reason for the warmup steps, as we require at least twice (and a rule of thumb suggests four times) more of them to stabilize the seasonality levels.

Epochs and sub-epochs

An epoch is usually defined as using all the training data once. We use a different definition here. It is based on the number of updates or processed batches. During training, we execute several time steps on the batch series. At each non-warmup step, we collect an error (loss), and at the end, we execute a single update based on the average of these errors. So, we execute a single update per batch. We aim to define the epoch as the number of updates that bring about a meaningful change to the system. Since the dataset contains a small number of series, a single scan through all of them would not be sufficient. Instead, we define a single epoch as actually composed of several “sub-epochs,” each defined traditionally as the use of all available series.

An additional factor is the batch size: as it increases, the number of updates per sub-epoch decreases, so the number of sub-epochs must also increase. However, in our experience, linear growth is too fast and risks overfitting within a single epoch. Therefore, we developed a sublinear formula, as shown in (19) of [D5].

8.7. Experiments and results

Table I in [D5] summarizes some average point forecast errors. The baseline models included classical statistical models (ARIMA and ES), several ML and NN models (LSTM, MTGNN, DeepAR, WaveNet, and N-BEATS), and a predecessor of the proposed model (ES-dRNNe). Clearly, the ES-dRNNe is the best model, and even a single, non-ensembled version of it is the second best.

To assess the PIs, we calculate the number of observed values in the PIs, below the PIs, and above the PIs. We achieved: $90.14\% \pm 2.43\%$, $4.88\% \pm 1.29\%$, and $4.98\% \pm 1.41\%$, for the expected 90% coverage, below the 5th percentile, and above the 95th percentile, respectively. Most of the benchmark algorithms were unable to provide the PIs, so we do not have a comparison.

8.8. Ablation study

The ablation study was quite broad; we removed many components from the best model, as shown in Section IIID of [D5].

As often happens, although the scale was larger than expected, feature engineering turned out to be the most important factor: the worst-performing model was the one that used only time series-derived inputs (i.e., no dates, etc.).

Embedding of dates was also essential – removing it produced the second-worst model. So, even a complicated RNN still benefits from good input preprocessing.

Seasonality levels as inputs appear very helpful (third worst) – note that in this case, ES is working fully; the series are deseasonalized and normalized, but the input does not “see” the seasonality coefficients. This was a surprise to me, but it was not a fluke; I have seen this behavior in other projects since then.

However, what was surprising was how strong the models were despite not having the full dilation mechanisms. I think the explanation is that the input window was large, spanning one week, so states delayed by a few days were not too important. Additionally, the weather, the main short-term driver, is challenging to predict over periods longer than 7-9 days, and therefore, past states are not that informative.

Moreover, finally, the standard LSTM performed competitively – well, in my experience dRNNCell is almost always better, but the margin differs from project to project.

8.9. Final thoughts

The proposed model had several improvements over its M4 Competition predecessor. Several mechanisms introduced here work reliably on many datasets, and with some modifications, I utilize them in my current work.

9. Paper [D6]: ES-dRNN with dynamic attention for short-term load forecasting

9.1. Overview

This paper was a follow-up to the paper described in the previous chapter. It introduced a new RNN cell, derived from the dRNNCell, which we called adRNNCell. The cell has its own mechanism to “concentrate” on the most important features. An RNN equipped with the new cells achieved an improvement in accuracy over the best reported in the previous paper.

9.2. Setup

The data, forecasting task, the loss function, pre- and post-processing, training details, and most of the model, including the ES component, but excluding the RNN, were as before; please refer to the previous paper.

9.3. RNN and adRNNCell

The graphs in Figs. 2 and 3 in [D6] compare the old dRNN and the new adRNN cells. The new adRNN cell can be seen as composed of two dRNN cells. The role of the first one is to produce a vector $\mathbf{m}_t$ that, after exponentiation, modulates the input $\mathbf{x}_t$ by element-wise multiplication, boosting or diminishing some part of it, before it is passed to the second dRNN cell that does the job of forecasting. Therefore, the second, forecasting dRNN operates on a modified input, and the adRNN cell can dynamically preprocess the input for the overarching goal of reducing the forecast error.

The RNN itself also underwent some changes compared to its previous layout, as shown in Fig. 4 of [D6]. Now we have three blocks, each with one cell, as hinted at as a possibility in the previous paper. However, the first layer consists of the adRNN cell, while the others are, as before, dRNN cells. This layout utilizes the adRNN's strength of dynamically preprocessing the input, while this ability is less important in later layers (although it takes additional processing time).

9.4. Results and conclusions

The result of the new method's ensemble, ES-adRNNe, is the best, as shown in Table I in [D6]. This is confirmed by the Giacomini-White test.

Dynamic modulation of the input, here achieved by using adRNNCell in the first layer, improves forecasting accuracy. A more often used alternative, Gated Linear Units (Dauphin, Fan, Auli, & Grangier, 2017), is simpler and faster, but less accurate, I believe.

10. Paper [D7]: Contextually enhanced ES-dRNN with dynamic attention for short-term load forecasting

10.1. Overview

This paper is the third one in the sequence dealing with short-term energy forecasting. A novelty here is the introduction of an additional RNN, whose task is to supply the main, forecasting, RNN with context information. This, and extending the training periods, made substantial improvements to the accuracy.

10.2. Setup

The dataset and forecasting task were the same as those in the previous paper. However, by using a more complex online preprocessing code that can handle missing values, we were able to extend the training period from 2016-2017, as in the two previous papers, to 2006-2017.

10.3. The context

ML models used for forecasting are almost always global models, trained on all series. However, when operating on a particular series, the inputs are usually derived only from this series' history and its external regressors. It is reasonable to assume that if we were able to enrich the input skillfully with information on what is happening with other series at the same time, we should be able to obtain a better forecast. One approach would be to find, somehow, for each series several “friendly” (context) series. Another approach would be to create an aggregated series that reflects the general trend in the dataset.

For large datasets, numbering thousands or more series, a brute-force way to find context series is unfeasible; instead, we could use, for example, geographical or product closeness. The problem, however, is that a similar series may not be invaluable. Generally, it is not easy to guess which series would be most helpful, and this must be done automatically and at scale. For aggregates, we could try to rely on some aggregates or representative series that make business or scientific sense; for example, by calculating an average and perhaps a few quantile series per geographical region. There are other possibilities that I have been working on, which I have not reported here; I believe they are a significant avenue for improvement in forecast quality.

However, even if we somehow find, for every series, a reasonably good set of context series, another problem appears: how to convert them into suitable inputs, considering that just concatenating them is unlikely to succeed, because in ML systems, inputs are position sensitive. Imagine that a positive trend in the first context series is predictive of a growing forecast of the target series, and a negative trend in the second context series is also predictive of a growing forecast. However, if we swap their places, it will not work.

The paper proposes an RNN-based forecasting system composed of two tracks: the first one, the context track, processes a representative subset of series and outputs an opaque vector

that, optionally after being modulated to suit a particular series, is concatenated with other features and becomes an input to the second, main forecasting track. The idea of context has been inspired by (Qin, et al., 2017), but the context here is calculated in a quite different, and much more computationally efficient way; notably, it does not use standard attention (it should also be noted that this approach is distinct from an entirely different attention mechanism used internally by adRNNCell).

10.4. Context and forecasting tracks

The pre- and post-processing is similar to one of the previous papers, but is extended, as we have two RNNs and two tracks, synchronized in time – see Fig. 1 in [D7]. Generally, as mentioned above, choosing a context series is a difficult task; however, in this case, it is easier because we have a small number of series, and many of them have significant gaps. Therefore, we decided to use all the series without missing values as context series, totaling 18. As to the positional stability, it is also not an issue here, because the context series are the same for all forecasted series, in the same positional sequence.

For every time step of the main RNN, the context RNN also takes it, processing a batch of 18 series. For each series, it outputs a small, opaque vector, which is finally flattened to a single context vector. However, it does not follow that each series of the main forecasting track receives the same context, because before the vector is appended to the input of the forecasting track, it is modified by element-wise multiplication with a per-series parameter. This parameter is initialized as a vector of ones but gets modified during the learning process. Again, the use of per-series parameters is a helpful trick when the number of series is small, as in this case.

10.5. Forecasting system

The forecasting track, apart from being enriched with context information, is very similar to that of the previous paper – see Fig. 5 in [D7]. Both RNNs also have three blocks with a single layer each; the only difference is that all cells are adRNN cells, not a mix of adRNNs and dRNNs, as we used only the more advanced adRNN cells. We used the same loss function.

The model development and hyperparameter search were executed by splitting the data into training (years 2006–2015) and validation (2016–2017) sets. Generally, the hyperparameters were chosen to minimize the validation forecasting error, but only after achieving a close to zero bias in the forecast. Then, using the best hyperparameters, final training was done on data from 2006 to 2017, and the test was performed on 2018 data. This schedule, which involved a significantly more extended training period, proved to be highly beneficial. However, when it comes to the baseline (comparison) models, some of them were unable to deal with missing values, so they could not be trained on the extended period, and some actually got better results on the shorter training period, as used in the previous paper. In such cases, we report their names with an asterisk in Table 2 in [D7].

10.6. Results

As shown in Table 2 of [D7], the new model, cES-adRNN, even in its individual (non-ensemble) version, produces more accurate forecasts than its ensembled predecessor (ES-AdRNN100) and all baseline models.

It is interesting to observe in Fig. 8 in [D7] how the accuracy depends on the ensemble size: there is little to be gained by increasing the ensemble beyond approximately 20 members.

10.7. Ablation study

We tried several reduced variants of the model; the conclusion was that the extended training period was the most important factor (MAPE reduced by 0.29%). The second most important factor was the addition of the context track (MAPE reduced by 0.18%).

The magnitude of the impact of the training period length was unexpected, because we are dealing here with a short-term forecasting task. This result nevertheless illustrates that the ML methods, and NNs in particular, thrive in data-rich environments.

10.8. Conclusion

Global forecasting models, although trained across many series, are likely to benefit from additional contextual information. In this paper, despite the small size of the dataset, we were able to provide a helpful context and, through per-series modifiers, efficiently customize it for each series. How to do it in larger datasets is an interesting and important topic for future work.

11. Paper [D8]: Any-quantile probabilistic forecasting of short-term electricity demand: Fusing uncertainties from diverse sources

11.1. Overview

In this paper, we continued working on high-frequency and short-term (48 hours ahead) energy demand forecasting; however, instead of predicting only the expected value and a small number of quantiles (which were largely ignored in the result analysis in previous papers), we explored the development of systems that learn to forecast at any quantile level requested at serving/inference time – thus enabling truly probabilistic forecasting.

The short-term electric energy demand contains a lot of unpredictability, as it is driven by weather, equipment maintenance and failures, as well as some sports events, among other factors. Consequently, a probabilistic forecast offers significant value for applications such as short-term planning, scheduling optimization, and energy trading.

11.2. Main innovations

- A simple any-quantile meta architecture.
- Using Beta distribution for sampling quantile levels.

11.3. Methodology

The methodology is broadly based on (Gouttes, Rasul, Koren, Stephan, & Naghibi, 2021) but with some simplifications and extensions. Our approach is illustrated in Fig. 1 of [D8]: during training, the NN receives, in addition to the standard input, a randomly generated quantile level q (between 0 and 1). The loss function, based on the pinball loss, is also calculated for the chosen quantile level. During prediction, a desired q is passed.

This concept is applicable across various types of forecasting systems. This paper reports on two models: my approach, based on ES-RNN, and Oreshkin’s approach, based on N-BEATS. We compared our any-quantile approach with a diverse collection of ML forecasting models (from the GluonTS library) and with classical statistical algorithms configured to produce quantile forecasts.

11.4. Research questions

My ES-RNN and Oreshkin’s NBEAT are high-accuracy algorithms proven to work well in many circumstances; however, their architectures differ significantly. We wanted to check how well they will work in the new setting and identify any necessary or helpful modifications. For example, the original idea paper (Gouttes, Rasul, Koren, Stephan, & Naghibi, 2021) used a separate Implicit Quantile Network, which took an embedded quantile level as input and modified the standard (point forecast) output of the NN. We instead decided to use a conceptually more straightforward approach where a single NN takes the quantile level together

with the other inputs. Moreover, the quantile level itself can be provided in several different ways, and we aim to identify which approach works best.

The performance of other ML algorithms, as well as statistical algorithms evaluated using density metrics, was also of interest. I will concentrate on my algorithm: AQ-ESRNN.

11.5. AQ-ESRNN

AQ-ESRNN is derived from ES-dRNN [D6] and illustrated in Fig. 2 in [D8]. The preprocessing, postprocessing, and general architectural approach of AQ-ESRNN and ES-dRNN are similar.

RNN

We use a deep residual RNN. The RNN has several inputs, including the randomly generated quantile level q and a normalized, deseasonalized, and squashed vector of past values $w = 168$. Additionally, the input includes: (i) date features $\mathbf{d}_t^w \in \{0, 1\}^7$, $\mathbf{d}_t^m \in \{0, 1\}^{31}$ and $\mathbf{d}_t^y \in \{0, 1\}^{52}$, i.e., binary one-hot vectors encoding day of the week, day of the month and week of the year, respectively; (ii) seasonality components, $\hat{\mathbf{s}}_t$ i.e., a vector of 48 seasonal components, and (iii) logarithm of the normalizer, $\log_{10}(\bar{y}_t)$.

The input is fed to the deep dilated RNN architecture consisting of five recurrent layers as shown in Fig. 2. Note that more layers are used here compared with the three layers in [D6], even though only 48 hourly values were forecasted, not 24·5 for the expected value plus four quantile levels, as before. This indicates that forecasting any quantile is more challenging.

The RNN employs attentive dilated RNN cells, adRNNCells, introduced in [D6]. They have a receptive field whose size is determined by a dilation parameter ($d \geq 1$). Larger dilations are assigned to top layers, i.e., $d = 2, 4, 7, 14, 28$. The dilations are in day units, as we step in time by 24 h . Residual connections are added to flow around each adRNNCell to avoid the vanishing gradient problem. The quantile level q is appended to every layer, including the last, linear one. We also found it beneficial to pass all other inputs to every cell, although the ablation studies showed that the improvement was relatively small.

Postprocessing

The postprocessing, as usual, “unwinds” the formulas used for the time series preprocessing, as seen in (14) in [D8]. We also found that an additional postprocessing of the RNN forecast via quantile sorting noticeably improved distributional forecasting accuracy. For every horizon length (1 to 48), we sort the forecasted values for all quantiles.

Training methodology

We use the standard quantile loss function (6), operating on normalized values. A unique quantile level q is sampled randomly for each time series in the batch. We found that AQ-ESRNN training benefits from non-uniform sampling of q , with emphasis on the tails. We

attribute this to the fact that ESRNN uses saturating activation functions, which lead to diminishing gradients at the tail of the distribution. It is also more challenging to fit very small (close to 0) and very large (close to 1) quantile levels well, because the quantile function is likely to change quickly in those regions. However, these edge quantile levels are often very important for planning purposes. So, we decided to oversample in the tails. In particular, we use a symmetric Beta distribution with both shape parameters less than 1, e.g., $\text{Beta}(0.3, 0.3)$ – see Fig. 3 in [D8]. Interestingly, the AQ-NBEATS system did not need oversampling – probably because it uses ReLUs, not squashing functions like sigmoid and tanh.

11.6. Metrics

Our primary probabilistic forecast error metric was the Continuous Ranked Probability Score (CRPS), defined in Equation (1) of [D8]. However, the CRPS is also equal to the doubled average pinball loss, and this is the form we used in our work – we approximated the integral by sampling or dense covering (e.g., over 100 steps) of the 0-1 interval of q . Additionally, we used the normalized CRPS (N-CRPS), given in (18) in [D8].

The CRPS is, in the framework of (Gneiting & Raftery, 2007), a measure of *sharpness*. A forecast should also be calibrated, meaning that the average Relative Frequency, defined as the average number of cases in which the actual value falls below the predicted quantile:

$$RF(g) = \frac{1}{M} \sum_{i=1}^M \mathbb{1}\{y_i \leq \hat{y}_{q,i}\}$$

should be close to the quantile level q . We defined the average absolute deviation of it as MARFE – Mean Absolute Relative Frequency Error, as seen in (20) in [D8], calculated on a grid of q values $\{0.001, 0.01, 0.02, \dots, 0.99, 0.999\}$. This was our primary *calibration* metric.

11.7. Results

My algorithm performed slightly better than AQ-NBEATS, and both were superior to the other methods, as measured by CRPS (see Table 3 in [D8]). So, when it comes to *sharpness*, the two main algorithms, AQ-ESRNN and AQ-NBEATS, were clearly ahead.

However, when it comes to coverage and MARFE, some benchmark algorithms, such as QR (Quantile Linear Regression), performed very well, as shown in Figs. 8 and 9 in [D8]. The winner here was a lowly QR from the `sklearn-linear_model` Python library. My results were worse, primarily due to the behavior near the edges, e.g., for 98th percentile coverage. Both low and high quantiles were incorrect: the 1st percentile forecast tended to be too high, and the 99th percentile forecast tended to be too low. So, it appears that despite oversampling, the algorithm still has problems at the edges of the q values. This is something we hope to correct in the following paper.

11.8. Ablation studies

The results of the ablation study are listed in Table 4 in [D8]. Reducing the number of layers was clearly detrimental. We also experimented with removing the quantile level input from each layer (AQ-ESRNN-noQuantInMidLayers); the resulting NN, shown in Fig. 11a in [D8], exhibited a noticeable drop in accuracy. Further simplifying the structure by removing the NN input to each layer (Fig. 11b) further reduced accuracy, albeit on a smaller scale.

Replacing adRNN cells with their simpler predecessors, dRNNs, resulted in a substantial loss of accuracy, which was even more pronounced when standard LSTMs were used. The most striking result is the dramatic loss of accuracy when the system is run without sorting.

11.9. Some other points and conclusions

Sorting

Interestingly, AQ-NBEATS suffered almost no reduction in accuracy without sorting. Generally speaking, sorting is an acceptable method, e.g., it guarantees that the CRPS never decreases (Chernozhukov, Fernández-Val, & Galichon, 2010). However, it remains a puzzle why its impact differs so much between the two leading systems in this study.

Quantile injection

As mentioned in the Research Questions chapter, we decided not to use Implicit Networks, but instead to feed the quantile level directly to the NN. This can be done in several ways, the simplest being to append it to the input. My best-performing network also appended it to every layer. The same happened with AQ-NBEATS, although its best version also applied a linear (FiLM) layer for inputs, while I relied on the adRNN cells' native ability to strengthen or weaken attribute values. Additionally, my experiments did not yield any improvement when attempting to embed the q level into a vector, as was done in (Gouttes, Rasul, Koren, Stephan, & Naghibi, 2021).

11.10. Conclusion

Our modifications to (Gouttes, Rasul, Koren, Stephan, & Naghibi, 2021) were effective. We got a more straightforward and more flexible model. Both ESRNN and N-BEATS were successfully modified to perform any-quantile forecasting. The paper left some unsolved issues, including, in the case of AQ-ESRNN, a (too) heavy dependence on sorting, and in the case of both algorithms, too narrow prediction intervals for large coverage values. We have some solutions to these problems, which we hope to present in future papers.

12. Summary

Time series forecasting blends science, engineering, and some art. While model fashions shift, practical forecasting remains valuable, with competitions grounding the field and discouraging a focus on impractical theories. Neural networks offer numerous opportunities for innovation in preprocessing, algorithms, and training, and advances from other fields can often benefit forecasting; however, popular methods are sometimes overhyped.

My journey with time series forecasting began with learning about classical statistical algorithms, such as ARIMA, Exponential Smoothing, and Theta. I then built a family of nonlinear extensions to Exponential Smoothing (ES) models, fitted with Markov Chain Monte Carlo (MCMC) using the Bayesian Statistics tool Stan (Stan Development Team, 2015).

Then I discovered that dynamic computational graph NN tools, initially limited to Dynet (Neubig, et al., 2017), later expanded to include PyTorch, allow for great flexibility in model building, similar to or even better than those provided by Stan. This allowed me to build my M4 Competition-winning model, which utilized an LSTM RNN but intertwined with some exponential smoothing formulas. Using neural networks, the Exponential Smoothing formulas can be extended in a way that is not feasible with statistical models, e.g., using dynamic (changing with every time step) smoothing coefficients. The other important features of my M4 models were the use of a global model approach (training across all series), with some per-series parameters, and adaptive (on-the-fly) time series data preparation.

Data preparation is essential, notwithstanding the perennial yearning among Data Scientists to outsource it entirely to Deep Learning. It also highlights some differences between practitioners and academics: in business, a simple model operating on well-preprocessed data is a great solution, while among academics, only advanced architectures seem to be valued.

Custom loss functions are handy; they should reflect the forecasting goals, including sensitivity of the loss in particular situations, such as during holidays, the asymmetry between over- and under-forecasting, and so on. Generally, the loss should be a weighted combination of a statistical measure (such as mean absolute deviation) and business loss (e.g., storage costs or penalties for not delivering contracted goods). Sadly, in most papers, just the oversensitive-to-outliers squared loss is used. My experience has led me to conclude that a normalized pinball loss is probably the best statistical loss: by introducing “training quantiles,” one can achieve a good coverage, and by adjusting a central quantile until it yields zero bias, one can achieve a good approximation of expected value – all without using the squared loss.

LSTM is a powerful cell designed for processing sequences, making it well-suited for time series forecasting. I introduced several improvements, e.g., mixing delayed states and dynamically modifying the input. With dilated RNNs, it is also very useful to modify the training procedure by using “warmup” periods, where several forward steps are executed without calculating the error. The warmup should also be part of validation and testing.

Creating and modifying NN architectures is interesting; unfortunately, we too often see persistent attempts to bend neural network architectures successful in some areas into time series forecasting, whether it makes sense or not. I would categorize recent struggles to utilize Large Language Models (LLMs) in the latter category. LLMs were not designed and trained

for time series forecasting; the structure of text is quite different from a vector of time series numbers. However, there is an enormous scope for utilizing LLMs, and generally, Generative AI models, to extract indicators, e.g., market sentiment, and to convert text descriptions or images into useful external regressors.

Enriching the input associated with each series by some contextual information can also bring substantial benefits. This contextual information may, as mentioned above, be a global indicator, such as sentiment or inflation, but it may also reflect what is happening with other series at the same time in a dataset. Doing this becomes more difficult for larger datasets, but it is a challenge worth pursuing.

Risk of overtraining is ever-present when working with small and medium-sized datasets. It can be delayed by adopting a scheme of random sampling the beginning of the time steps and stopping after a number of steps. This also allows us to train RNNs on very long series.

Ensembling is always worth doing; even simple ensembles of identical models, randomly initialized, using a straightforward aggregation function like the median or mean, bring the benefits of improved accuracy and stability. Time series forecasting is seldom a significant data problem; it is usually not challenging to marshal resources for a small (3-9 members) ensemble. Moreover, of course, one can go further by building teams of specialists and dynamically choosing a subset of them when forecasting a particular series at a specific time step.

One of the most useful, yet seldom pursued, ideas is scenario forecasting, which can also be viewed as modifying the output of a data-driven forecasting system with business expertise. It goes beyond providing a few quantile forecasts. I hope to contribute to it in the future. Moreover, of course, I am not immune to the allure of finding new NN architectures, perhaps more biologically inspired.

I want to thank Prof. Grzegorz Dudek and Dr. Pawel Pelka for encouraging me to write this thesis, providing feedback, and some help in editing.

References

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*.
- Bandara, K., Bergmeir, C., & Smyl, S. (2020). Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach. *Expert Systems with Applications*, 140, p. 112896.
- Ben-Ari, I., & Shwartz-Ziv, R. (2017). Sequence Modeling Using a Memory Controller Extension for LSTM. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, Y., Maddix, D., Turkmen, C., . . . Januschowski, T. (2022). Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Computing Surveys*, 55(6), pp. 1–36.
- Bergmeir, C. (2024). LLMs and Foundational Models: Not (Yet) as Good as Hoped. *Foresight: The International Journal of Applied Forecasting*, 73, pp. 33–38.
- Bergmeir, C. (2024b). *Fundamental limitations of foundational forecasting models*. Retrieved 12 30, 2025, from Invited talk given at the 2024 NeurIPS workshop “Time Series in the Age of Large Models”: <https://cbergmeir.com/talks/neurips2024/>
- Bergmeir, C., & Smyl, S. (2023). *ISF 2023 Practitioner Talks*. Retrieved 12 30, 2025, from <https://www.youtube.com/watch?v=KApC-TicQLo>
- Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT)*, (pp. 144–152).
- Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., . . . Huang, T. S. (2017). Dilated Recurrent Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 30.
- Chen, H., Ng, E., Smyl, S., & Steininger, G. (2024). Predicting Customer Lifetime Value Using Recurrent Neural Net. *arXiv preprint arXiv:2412.20295*.
- Chernozhukov, V., Fernández-Val, I., & Galichon, A. (2010). Quantile and probability curves without crossing. *Econometrica*, 78(3), pp. 1093–1125.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (pp. 1724–1734).
- Corradini, F., Gerosa, F., Gori, M., Lucheroni, C., Piangerelli, M., & Zannotti, M. (2025). A Systematic Literature Review of Spatio-Temporal Graph Neural Network Models for Time Series Forecasting and Classification. *Neural Networks*, p. 108269.
- Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017). Language modeling with gated convolutional networks. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, (pp. 933–941).
- Dudek, G., Smyl, S., & Peřka, P. (2023). Recurrent neural networks for forecasting time series with multiple seasonality: a comparative study. In O. Valenzuela, F. Rojas, L. J. Herrera, H. Pomares, & I. Rojas, *Theory and Applications of Time Series Analysis* (pp. 171–185). Springer Nature Switzerland.
- Ekambaram, V., Jati, A., Dayama, P., Mukherjee, S., Nguyen, N., Gifford, W. M., . . . Kalagnanam, J. (2024). Tiny Time Mixers (TTMs): Fast Pre-trained Models for Enhanced Zero/Few-Shot Forecasting of Multivariate Time Series. *Advances in Neural Information Processing Systems* (NeurIPS)*, 37, pp. 74147–74181.
- Elman, J. (1990). Finding Structure in Time. *Cognitive Science*, 14(2), 179–211.

- Gneiting, T., & Raftery, A. E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, 359–378.
- Godaheewa, R., Bandara, K., Webb, G. I., Smyl, S., & Bergmeir, C. (2021). Ensembles of Localised Models for Time Series Forecasting. *Knowledge-Based Systems*, 233, p. 107518.
- Gouttes, A., Rasul, K., Koren, M., Stephan, J., & Naghibi, T. (2021). Probabilistic time series forecasting with implicit quantile networks. *arXiv preprint arXiv:2107.03743*.
- Gu, A., & Dao, T. (2024). Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *First Conference on Language Modeling (COLM)*.
- Gu, A., Goel, K., & Ré, C. (2022). Efficiently Modeling Long Sequences with Structured State Spaces. *arXiv preprint arXiv:2111.00396*.
- He, K., Zhang, X., Ren, S., & Jian, S. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, (pp. 1026-1034).
- Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent Neural Networks for Time Series Forecasting. *International Journal of Forecasting*, 37(1), pp. 388–427.
- Hinton, G., Osindero, S., & Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7), 1527–1554.
- Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1, pp. 278–282.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
- Hong, T., Xie, J., & Black, J. (2019). Global Energy Forecasting Competition 2017: Hierarchical Probabilistic Load Forecasting. *International Journal of Forecasting*, 35(4), pp. 1389–1399.
- Hu, M., & Root, H. E. (1964). An adaptive data processing system for weather forecasting. *Journal of Applied Meteorology (1962-1982)*, 513–523.
- Hyndman, R. (2018). *A brief history of time series forecasting competitions*. Retrieved 12 30, 2025, from Hyndsight blog: <https://robjhyndman.com/hyndsight/forecasting-competitions/>
- Hyndman, R., Koehler, A., Ord, K., & Snyder, R. (2008). *Forecasting with Exponential Smoothing*. Springer.
- Kim, J., El-Khamy, M., & Lee, J. (2017). Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition. *arXiv preprint arXiv:1701.03360*.
- Kim, J., Kim, H., Kim, H., & Lee, D. (2025). A comprehensive survey of deep learning for time series forecasting: architectural diversity and open challenges. *Artificial Intelligence Review*, 58(7), pp. 1–95.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, (pp. 1097–1105).
- Laptev, N., Yosinski, J., Li, E., & Smyl, S. (2017). Time-series Extreme Event Forecasting with Neural Networks at Uber. *Proceedings of the 34th International Conference on Machine Learning (ICML)*.
- LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural network*. MIT Press.
- Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2021). Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *International Journal of Forecasting*, 37(4), pp. 1748–1764.
- Loboz, C., Jia, H., & Smyl, S. (2012, 2). *United States Patent No. US8122117B2*.

- Loboz, C., Smyl, S., & Nath, S. (2010). DataGarage: Warehousing Massive Performance Data on Commodity Servers. *Proceedings of the VLDB Endowment*, (pp. 1447-1458).
- Long, X., Schmidt, D. F., Bergmeir, C., & Smyl, S. (2025). Fast Gibbs sampling for the local-seasonal-global trend Bayesian exponential smoothing model. *Statistics and Computing*, 35(3), p. 77.
- Makridakis, S., & Hibon, M. (2000). The M3-Competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4), 451–476.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). Predicting/hypothesizing the findings of the M4 Competition. *International Journal of Forecasting*, 36(1), pp. 29–36.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1), pp. 54–74.
- Meinshausen, N. (2006). Quantile Regression Forests. *Journal of Machine Learning Research*, 983–999.
- Merrill, W., Petty, J., & Sabharwal, A. (2024). The Illusion of State in State-Space Models. *Proceedings of the 41st International Conference on Machine Learning (ICML)*, (pp. 35492–35506).
- Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastasopoulos, A., . . . Kumar, G. (2017). DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*.
- Ng, E., Wang, Z., Chen, H., Yang, S., & Smyl, S. (2020). Orbit: Probabilistic Forecast with Exponential Smoothing. *arXiv preprint arXiv:2004.08492*.
- Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2022). A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. *arXiv preprint arXiv:2211.14730*.
- Oreshkin, B. N., Carpo, D., Chapados, N., & Bengio, Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *International Conference on Learning Representations (ICLR)*.
- Pełka, P., Dudek, G., & Smyl, S. (2022). An Ensemble of Attentive Recurrent Networks with Randomized Dilations for Forecasting. *Proceedings of the 3rd Polish Conference on Artificial Intelligence (PP-RAI)*.
- Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., & Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*.
- Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory Report No. 85–460–1.
- Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (pp. 318–362). MIT Press.
- Smyl, S. (2017). Ensemble of specialized neural networks for time series forecasting. *The 37th International Symposium on Forecasting (ISF)*.
- Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1), 75–85.
- Smyl, S., & Kuber, K. (2016). Data Preprocessing and Augmentation for Multiple Short Time Series Forecasting with Recurrent Neural Networks. *The 36th International Symposium on Forecasting (ISF)*.
- Smyl, S., & Zhang, Q. (2015). Fitting and Extending Exponential Smoothing Models with Stan. *The 35th International Symposium on Forecasting (ISF)*.

- Smyl, S., Bergmeir, C., Dokumentov, A., Long, X., Wibowo, E., & Schmidt, D. (2025). Local and global trend Bayesian exponential smoothing models. *International Journal of Forecasting*, 41(1), pp. 111–127.
- Smyl, S., Dudek, G., & Pelka, P. (2023). Forecasting Cryptocurrency Prices Using Contextual ES-adRNN with Exogenous Variables. *International Conference on Computational Science*, (pp. 450-464).
- Smyl, S., Ranganathan, J., & Pasqua, A. (2018). *M4 forecasting competition: Introducing a new hybrid es-rnn model*. Retrieved 12 30, 2025, from <https://eng.uber.com/m4-forecasting-competition>
- Sriramulu, A., Bergmeir, C., & Smyl, S. (2024). Context Neural Networks: A Scalable Multivariate Model for Time Series Forecasting. *arXiv preprint arXiv:2405.07117*.
- Stan Development Team. (2015). *Stan Modeling Language Users Guide and Reference Manual, Version 2.9.0*. Retrieved 12 30, 2025, from <http://mc-stan.org/>
- Stepnicka, M., & Burda, M. (2015). Computational Intelligence in Forecasting – The Results of the Time Series Forecasting Competition. *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, (pp. 1–8).
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, pp. 3104–3112.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*, (pp. 6000–6010).
- Wang, Z., Kong, F., Feng, S., Wang, M., Yang, X., Zhao, H., . . . Zhang, Y. (2025). Is Mamba Effective for Time Series Forecasting? *Neurocomputing*, 619, p. 129178.
- Wu, H., Hu, T., Liu, Y., Zhou, H., Wang, J., & Long, M. (2022). TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. *arXiv preprint arXiv:2210.02186*.
- Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2023). Are Transformers Effective for Time Series Forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9), pp. 11121–11128.

Papers



A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting

Slawek Smyl

Uber Technologies, 555 Market St, 94104, San Francisco, CA, USA



ARTICLE INFO

Keywords:

Forecasting competitions

M4

Dynamic computational graphs

Automatic differentiation

Long short term memory (LSTM) networks

Exponential smoothing

ABSTRACT

This paper presents the winning submission of the M4 forecasting competition. The submission utilizes a dynamic computational graph neural network system that enables a standard exponential smoothing model to be mixed with advanced long short term memory networks into a common framework. The result is a hybrid and hierarchical forecasting method.

© 2019 International Institute of Forecasters. Published by Elsevier B.V. All rights reserved.

1. Introduction

Over recent decades, neural networks (NNs) and other machine learning (ML) algorithms have achieved remarkable success in various areas, including image and speech recognition, natural language processing (NLP), autonomous vehicles and games (Makridakis, 2017), among others. The key to their success is the fact that, given a large representative dataset, ML algorithms can learn to identify complex non-linear patterns and explore unstructured relationships without hypothesizing them a priori. Thus, ML algorithms are not limited by assumptions or pre-defined data generating processes, which allows the data to speak for itself.

However, the superiority of ML is not apparent when it comes to forecasting. While ML algorithms have been successful (Weron, 2014) in some applications like energy forecasting (Dimoukas, Mazidi, & Herre, 2019), where the series being extrapolated are often numerous, long, and accompanied by explanatory variables, the performance of ML algorithms in more typical time series forecasting, where the data availability is often limited and regressors are not available, tends to be below expectations (Makridakis, Spiliotis, & Assimakopoulos, 2018b).

None of the popular ML algorithms have been created for time series forecasting, and time series data need to be preprocessed in order for them to be used for forecasting.

The strength of ML algorithms, and in fact the requirement for their successful use, is cross-learning, i.e., using many series to train a single model. This is unlike standard statistical time series algorithms, where a separate model is developed for each series. However, the preprocessing needs to be thought over well in order to learn across many time series. NNs are particularly sensitive in this area, as will be expanded later.

There are many good rules regarding preprocessing, but it remains an experiment-intensive art. One of the most important ingredients in the success of this method in the M4 Competition was the on-the-fly preprocessing that was an inherent part of the training process. Crucially, the parameters of this preprocessing were being updated by the same overall optimization procedure (stochastic gradient descent) as weights of the NNs, with the overarching goal of accurate forecasting (minimizing forecasting errors).

The preprocessing parameters were actually those from (slightly simplified) updating formulas of some models from the exponential smoothing family. Thus, what is presented here is a hybrid forecasting method that mixes an exponential smoothing (ES) model with advanced long short term memory (LSTM) neural networks in a common framework. The ES equations enable the method to capture the main components of the individual series, such as seasonality and level, effectively, while the LSTM networks allow non-linear trends and cross-learning. In this regard, the data are exploited in a

E-mail address: slaweks@hotmail.co.uk.

hierarchical manner, meaning that both local and global components are utilized in order to extract and combine information at either a series or a dataset level, thus enhancing the forecasting accuracy.

The rest of the paper is organized as follows. Section 2 introduces the method and describes it in a general sense, while Section 3 gives more implementation details. Section 4 concludes by sketching some general modelling possibilities that are provided by recent NN systems and probabilistic programming languages, and, in this context, traces back the development of the models described in this paper.

2. Methodology

2.1. Intuition and overview of the hybrid method

The method effectively mixes ES models with LSTM networks, and in so doing, provides forecasts that are more accurate than those generated by either pure statistical or ML approaches, thus exploiting their advantages while avoiding their drawbacks. This hybrid forecasting approach has three main elements: (i) deseasonalization and adaptive normalization, (ii) generation of forecasts and (iii) ensembling.

The first element is implemented with state space ES-style formulas. The initial seasonality components (e.g. four for the quarterly series) and smoothing coefficients (two of them in the case of a single seasonality system) are per-series parameters and were fitted, together with global NN weights, by stochastic gradient descent (SGD). Knowing these parameters and the values of the series allows the seasonality components and levels to be calculated, and these are used for deseasonalization and normalization. The deseasonalization of seasonal series was very important in the M4 Competition, given that the series were provided as numeric vectors without any time stamp, so that there was no way of incorporating calendar features like the day of the week or the month number. Also, the series came from many sources, so their seasonality patterns varied.

The second element is a NN that operates on deseasonalized and normalized data, providing the horizon-steps ahead (e.g. 18 points in case of monthly series) outputs that were subsequently re-normalized and re-seasonalized to produce forecasts. The NN is global, learning across many time series.

The final element of the method is the ensembling of the forecasts made in the previous step. This includes ensembling the forecasts produced by the individual models from several independent runs, sometimes produced by a subset of concurrently-trained models, and also averaging those generated by the most recent training epochs. This process enhances the robustness of the method further, mitigating the model and parameter uncertainty (Petroopoulos, Hyndman, & Bergmeir, 2018) while also exploiting the beneficial effects of combining (Chan & Pauwels, 2018).

Based on the above, it can be said that the method has the following two special characteristics:

- It is hybrid, in the sense that statistical modeling (ES models) is combined concurrently with ML algorithms (LSTM networks).
- It is hierarchical in nature, in the sense that both global (applicable to large subsets of all series) and local (applied to each series individually) parameters are utilized in order to enable cross-learning while also emphasizing the particularities of the time series being extrapolated.

2.2. Method description

2.2.1. Deseasonalization and normalization

The M4 time series, even within the same frequency subset, e.g. monthly, come from many different sources and exhibit a range of seasonality patterns. In addition, the starting dates of the series are not provided. In such circumstances, the NNs are unable to learn how to deal with seasonality effectively. A standard remedy is to apply a deseasonalization at preprocessing time. This solution is adequate but not ideal, as it separates the preprocessing from the forecasting completely, and the quality of the decomposition is likely to be worst near the end of the series, where it counts most for the forecast. One can also observe that the deseasonalization algorithms, however sophisticated and robust, were not designed to be good preprocessors for NNs. Classic statistical models, such as those from the exponential smoothing family, show a better way: the forecasting model has integral parts that deal with the seasonality.

In most NN variants, including LSTMs, the update size of weight w_{ij} is proportional to the final error, but also to the absolute value of the strength of the associated signal (coming from neuron i in the current layer to neuron j in the next layer). Thus, the NNs behave like analogue devices, even if implemented on a digital computer: small-valued inputs will have small impacts during the learning process. Normalizing each series globally to an interval like $[0-1]$ is also problematic, as the values to be forecast may lie outside this range, and, more importantly, for series that change a lot over their lifespan, the parts of the series with small values will be ignored. Finally, information on the strength of trend is lost: two series of the same lengths and similar shapes, but one growing from 100 to 110 and another from 100 to 200, will look very similar after the $[0-1]$ normalization. Thus, while normalization is necessary, it should be adaptive and local, where the “normalizer” follows the series values.

2.2.2. Exponential smoothing formulas

Keeping in mind that the M4 series all have positive values, the models of Holt (Gardner, 2006) and Holt and Winters (Hyndman, Koehler, Ord, & Snyder, 2008) with multiplicative seasonality were chosen. However, these were simplified by the removal of the linear trend: the NN was tasked to produce a trend that was most likely to be non-linear. Moreover, non-seasonal (yearly and daily data), single-seasonal (monthly, quarterly, and weekly data) or double-seasonal (hourly data) models (Taylor, 2003) were used, depending on the frequency of the data. The updating formulas for each case are as follows:

Non-seasonal models:

$$l_t = \alpha y_t + (1 - \alpha)l_{t-1} \quad (1)$$

Single seasonality models:

$$\begin{aligned} l_t &= \alpha y_t / s_t + (1 - \alpha)l_{t-1} \\ s_{t+K} &= \beta y_t / l_t + (1 - \beta)s_t \end{aligned} \quad (2)$$

Double seasonality models:

$$\begin{aligned} l_t &= \alpha y_t / (s_t u_t) + (1 - \alpha)l_{t-1} \\ s_{t+K} &= \beta y_t / (l_t u_t) + (1 - \beta)s_t \\ u_{t+L} &= \gamma y_t / (l_t s_t) + (1 - \gamma)u_t, \end{aligned} \quad (3)$$

where y_t is the value of the series at point t ; l_t , s_t , and u_t are the level, seasonality, and second seasonality components, respectively; K is the number of observations per seasonal period, i.e., four for quarterly, 12 for monthly and 52 for weekly; and, finally, L is the number of observations per second seasonal period (for hourly data, 168). Note that s_t and u_t are always positive, while the smoothing coefficients α , β and γ take a value between zero and one. These restrictions can be implemented easily by applying $\exp()$ to the underlying parameters of the initial seasonality components and $\text{sigmoid}()$ to the underlying parameters of the smoothing coefficients.

2.2.3. On-the-fly preprocessing

The above formulas allow the level and seasonality components to be calculated for all points of each series. These components are then used for deseasonalization and adaptive normalization during the on-the-fly preprocessing. This step is a crucial part of the method and is described in this section.

Each series was preprocessed anew for each training epoch, because the parameters (initial seasonality components and smoothing coefficients) and the resulting levels and seasonality components were different during each epoch.

The standard approach of constant size, rolling input and output windows was applied, as is shown in Fig. 1 for the case of a monthly series. The size of the output window was always equal to the forecasting horizon (e.g., 13 for the weekly series), while the size of the input window was determined by a rule that, for seasonal series, it should cover at least a full seasonal period (e.g., being equal to or larger than four in the case of quarterly series), while for non-seasonal series the size of the input window should be close to the forecasting horizon. However, the exact size was defined after conducting experimentation (backtesting). Please note that, unlike in many other recurrent neural network (RNN) based sequence processing systems, the input size is larger than one. This works better because it allows the NN to be exposed to the immediate history of the series directly.

Preprocessing was rather simple: at each step, the values in the input and output windows were normalized by dividing them by the last value of the level in the input window (the thick blue dot on Fig. 1), and then, in the case of seasonal time series, divided further by the relevant seasonality components. That resulted in the input and output values being close to one, irrespective of

the original amplitude of the series and its history. Finally, a squashing function, $\log()$, was applied. The squashing function prevented outliers from having an unduly large and disturbing effect on the learning.

In addition, the domain of the time series (e.g. finance or macro) was one-hot encoded as a six-long vector and appended to the time series derived features. The domain information was the only meta information available and I considered it prudent to expose the NN to it.

It is generally worthwhile to increase the size of the input window and extract more sophisticated features, like the strength of the seasonality or the variability, when preprocessing for NNs, but such approaches were not adopted here for several reasons. The most important one was that many series were too short to afford a large input window, meaning that they could not be used for backtesting. Another reason was that creating features that summarize the characteristics of the series effectively, irrespective of their length, is not straightforward. It was only after the end of the competition that a promising R package called *tsfeatures* came to my attention (Hyndman, Wang, & Laptev, 2015; Kang, Hyndman, & Smith-Miles, 2017).

2.2.4. Forecast by NNs

As explained above, the NNs operated on deseasonalized, adaptively normalized, and squashed values. Their output needed to be “unwound”, in following way:

For non-seasonal models:

$$\hat{y}_{t+1..t+h} = \exp(\text{NN}(x)) * l_t \quad (4)$$

For single seasonality models:

$$\hat{y}_{t+1..t+h} = \exp(\text{NN}(x)) * l_t * s_{t+1..t+h} \quad (5)$$

For dual seasonality models:

$$\hat{y}_{t+1..t+h} = \exp(\text{NN}(x)) * l_t * s_{t+1..t+h} * u_{t+1..t+h}, \quad (6)$$

where x is the pre-processed input (a vector), $\text{NN}(x)$ is an NN output (a vector), l_t is the value of level at time t (last known data point) and h is the forecasting horizon. All operations are elementwise. The above is summarized in Fig. 2.

Note that the final forecast is actually an ensemble of many such forecasts, a procedure which is explained later in the paper.

2.2.5. Architectures of neural networks

In order to better understand the implementation, it is useful to classify the parameters of forecasting systems into the following three groups:

- **Local constants:** These parameters reflect the behavior of a single series; they do not change as we step through that series. For example, the smoothing coefficients of the ES model, as well as the initial seasonal components, are the local non-changing (constant) parameters.
- **Local states:** These parameters change as we step through a series, evolving over time. For instance, the level and seasonal components, as well as a recurrent NN state, are local states.

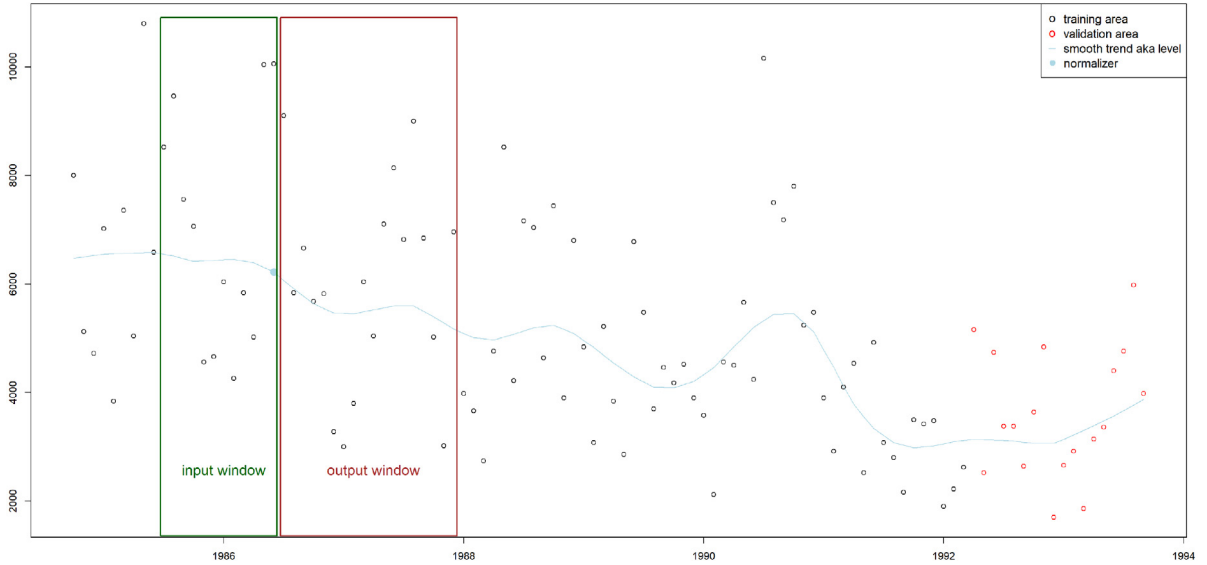


Fig. 1. An example showing how rolling windows are used for preprocessing a random monthly series. The last step is used for validation.

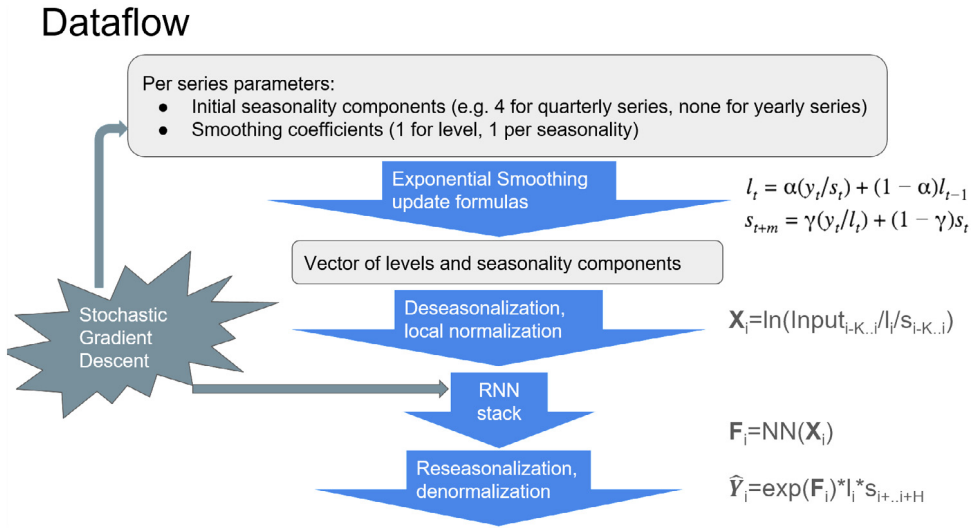


Fig. 2. Data flow and system architecture for the single seasonality case. X_i is the normalized, deseasonalized, and squashed input to the NN. F_i is the NN output. $\hat{Y}_i$ is the forecast, covering outputs $i + 1 \dots i + H$, where H is the forecasting horizon. l_i is scalar, the last level in the input window. X_i , $\hat{Y}_i$ and F_i are vectors. Ensembling is not shown.

- **Global constants:** These parameters reflect the patterns learned across large sets of series and are constant; they do not change as we step through a series. For example, the weights used for the NN systems are global constants.

Typical statistical time series methods are trained on individual series, meaning that they involve only local constant and local state parameters. On the other hand, standard ML methods are usually trained on large datasets, involving only global parameters. The hybrid method described here uses all three types of parameters, being partly global and partly time series specific. This type of modeling becomes possible through the use

of dynamic computation graph (DCG) systems, such as DyNet (Neubig, Dyer, Goldberg, Matthews, Ammar, Anastopoulos, et al., 2017), PyTorch (Paszke, Gross, Chintala, Chanan, Yang, DeVito, et al., 2017) and TensorFlow in “eager mode” (Abadi, Agarwal, Barham, Brevdo, Chen, Citro, et al., 2015). The difference between static and dynamic computational graph systems is that the latter have the ability to recreate the computational graph (built behind the scenes by the NN system) for each sample, here, for each time series. Thus, each series may have a partially unique and partially shared model.

The architecture deployed was different for each frequency and output type (point forecast or prediction intervals).

At a high level, the NNs of the model are dilated LSTM-based stacks (Chang, Zhang, Han, Yu, Guo, Tan, et al., 2017), sometimes followed by a non-linear layer and always followed by a linear “adapter” layer, the objective of which is to adapt the size of the state of the last layer to the size of the output layer (the forecasting horizon, or twice the forecasting horizon in case of prediction interval (PI) models). The LSTM stacks are composed of a number of blocks (here 1–2). In case of two (and theoretically more) blocks, the output of a block is added to the next block’s output using Resnet-style shortcuts (He, Zhang, Ren, & Sun, 2015). Each block is a sequence of one to four layers, belonging to one of the three types of dilated LSTMs: standard (Chang et al., 2017), with an attention mechanism (Qin, Song, Chen, Cheng, Jiang, & Cottrell, 2017) and a special residual version (Kim, El-Khamy, & Lee, 2017).

Dilated LSTMs use as part of their input the hidden state from previous, but not necessarily the latest, steps. In standard LSTMs and related cells, part of the input at a time t is the hidden state from step $t - 1$. In a cell that is k -dilated, e.g. three, the hidden state is taken from step $t - k$, so here $t - 3$. This improves long-term memory performance. As is customary for dilated LSTMs (Chang et al., 2017), they were deployed in stacks of cells with increasing dilations. Similar blocks of standard, non-dilated LSTMs performed slightly worse. Even bigger drops in performance would have happened if the recurrent NNs had been replaced with non-recurrent ones, indicating that the RNN state is useful for dealing with the time series and sequences more generally.

The general idea of the recurrent NN attention mechanism is that, instead of using the previous hidden state as in standard LSTMs, or the delayed state as in the case of dilated LSTMs, one calculates weights that are applied to a number of past hidden states in order to create an artificial weighted average state. This allows the system to “concentrate on” or “attend to” a particular single state or group of past states dynamically. My implementation is an extension of the dilated LSTM, so the maximum look-behind horizon is equal to the dilation. In the case of weekly series, the network consisted of single block with two layers, encoded as attentive (1,52). The first layer dilation is equal to one, so it is a standard LSTM, but the second layer calculates weights over the past 52 hidden states (as they become available, so at point 53 or later when stepping through a series). The weights are calculated using a separate standard two-layer NN that is embedded into the LSTM; its inputs are concatenations of the LSTM input and the last hidden state, and its weights are adjusted by the same gradient descent mechanism that operates on all other parameters.

Fig. 3 shows three examples of configurations; the first one generates point forecasts (PFs) for the quarterly series, the second one PFs for the monthly series, and the third one prediction intervals (PIs) for the yearly series.

- (a) The NN consists of two blocks, each one involving two dilated LSTMs, that are connected by a shortcut around the second block. The final element is the “adapter layer”; it is just a standard linear layer (the

transfer function equals identity) that adapts the hidden output from the fourth layer (the one with dilation = 8), usually 30–40 long, into the expected output size (here eight).

- (b) The NN consists of a single block composed of four dilated LSTMs, with residual connections as per (Kim et al., 2017). Please note that the shortcut arrows point correctly into the inside of the residual LSTM cell; this is a non-standard residual shortcut.
- (c) The NN consists of a single block consisting of two dilated LSTMs with the attention mechanism, followed by a dense non-linear layer (with $\tanh()$ activation), then by a linear adapter layer of the double size of the output, so that forecasts of both lower and upper bounds are generated simultaneously. The attention mechanism (Qin et al., 2017) slows the calculations considerably, but occasionally appeared best.

Later, I provide a table that lists architectures and hyperparameters for all cases, not just these three. Please keep in mind that, while the graph shows only the global parts of the models, the per-series parts are equally important.

3. Implementation details

This section provides more implementation details regarding the hybrid method. This includes information about the loss function, the hyperparameters of the models, and the ensembling procedures.

3.1. Loss function

3.1.1. Point forecasts

The error measure used in the M4 Competition for the case of the PFs was a combination of the symmetric mean absolute error (sMAPE) and the mean scaled error (MASE) (Makridakis, Spiliotis, & Assimakopoulos, 2018a). The two metrics are quite similar in nature, in the sense that both are normalized absolute differences between the predicted and actual values of the series. Recalling that the inputs to the NN in this system are already deseasonalized and normalized, I postulated that the training loss function does not need to include normalization: it could be just a simple absolute difference between the target values and the predicted ones. However, it became apparent during backtesting that the models tend to have a positive bias, probably as a result of applying a squashing function, $\log()$, to time series derived inputs and outputs to the NN. The system learned in the log space, but the final forecast errors are calculated back in the linear space. To counter this, a pinball loss with a τ value a bit smaller than 0.5 (typically 0.45–0.49) was used. The pinball loss is defined as follows:

$$L_t = (y_t - \hat{y}_t)\tau, \text{ if } y_t \geq \hat{y}_t \\ = (\hat{y}_t - y_t)(1 - \tau), \text{ if } \hat{y}_t > y_t. \quad (7)$$

Thus, the pinball function is asymmetric, penalizing actual values that are above and below a quantile differently so as to allow the method to deal with the bias. It is an important loss function on its own; minimizing

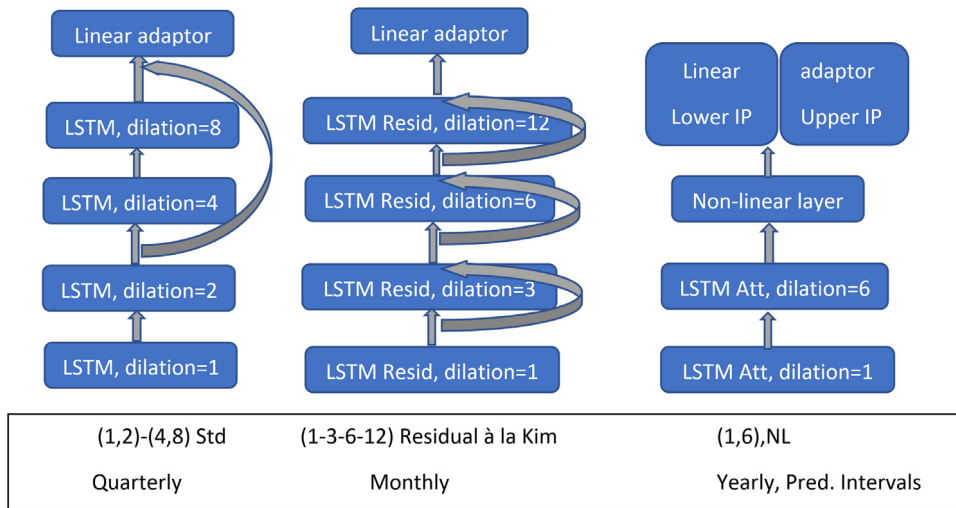


Fig. 3. NN architectures used for generating some of the PFs and PIs.

it produces quantile regression (Takeuchi, Le, Sears, & Smola, 2006).

3.1.2. Prediction intervals

The pinball loss function could have been adopted for generating the PIs as well. The requested coverage was 95%, so one could have tried to forecast 2.5% and 97.5% intervals. However, the competition metric for PIs was not based on separate upper and lower pinball losses; instead, it was a single formula called the mean scaled interval score (MSIS) (Makridakis et al., 2018a). Once again, the denominator of the MSIS was omitted, since the input to the NNs was already deseasonalized and normalized. It should be noted that, although the method provided the most precise PIs among those of all methods submitted, the positive bias mentioned above can still be observed for the case of the PIs, with the upper interval being exceeded less frequently than the lower one.

At this point I would like to draw the reader's attention to the great practical feature of NN-based systems: the ease of creating a loss function that is aligned with business/scientific objectives. For this application, the loss functions were aligned with the accuracy metrics used in the M4 Competition.

3.1.3. Level wiggleness penalty

Intuitively, the level should be a smooth version of the time series, with no seasonality patterns. One would expect this to be of secondary importance, and more of an aesthetic-level requirement. However, it turns out that the smoothness of the level influenced the forecasting accuracy substantially. It appears that when the input to the NN was smooth, the NN concentrated on predicting the trend, instead of over-fitting on some spurious, seasonality-related patterns. A smooth level also means that the seasonality components absorbed the seasonality properly. In functional data analysis, an average of squares of second derivatives is a popular penalty against the wiggleness of a curve (Ramsay & Silverman, 2002). However, such a penalty may be too strict and not robust enough

when applied to time series with occasional large shifts. In this regard, a modified version of this penalty was applied, which was calculated as follows:

- Calculate log differences, i.e., $d_t = \log(y_{t+1}/y_t)$, where y_t is point t of the series;
- Calculate differences of the above: $e_t = d_{t+1} - d_t$;
- Square and average them for each series.

This penalty, multiplied by a constant parameter in the range of 50–100, called the level variability penalty (LVP), was added to both PFs and PIs loss function. The level wiggleness penalty affected the performance of the method significantly, and it is conceivable that this submission would not have won the M4 Competition without it.

3.2. Ensembling and data subsetting

Two models (one for PFs and one for PIs) were built for each of the six single-frequency subsets (daily, weekly etc.). Each of the models was actually an ensemble at several levels, which are presented below.

3.2.1. Independent runs

A single run involves a full training of the models as well as the generation of forecasts for all series in the subset. However, each run for a given series produces a slightly different forecast, since the parameter initializations are random. Ensembling models constructed from different runs can mitigate the effect of randomness and decrease the uncertainty. Backtesting indicated that increasing the number of runs above the 6–9 range did not improve the forecasting accuracy, and as a result, the number of independent runs was limited accordingly.

3.2.2. Ensemble of specialists or simple ensemble

When it was computationally feasible, as turned out to be the case for all except the monthly and quarterly series, several concurrently-trained models, learning from different subset of series, were used rather than training

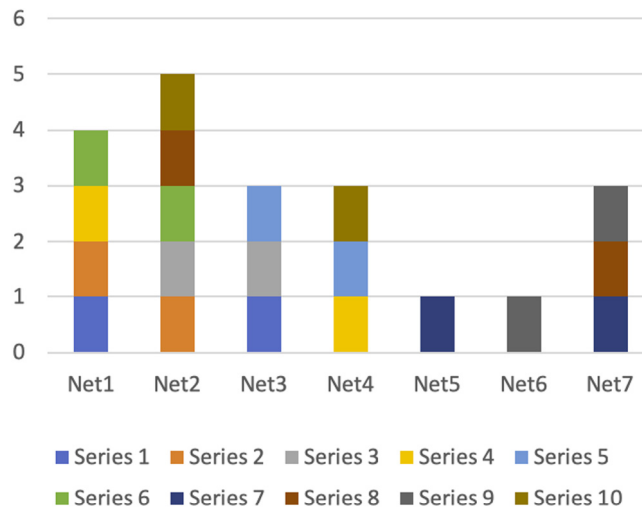


Fig. 4. An example allocation performed by the ensemble of specialists algorithm for a set of ten series to seven models, using the top two models per series.

a single model. This approach, called “ensemble of specialists”, was proposed originally by Smyl (2017), and is summarized below.

The main idea is that, when a dataset contains a large number of series from unknown sources, it is reasonable to assume that these could possibly be grouped in subsets, such that the overall forecasting accuracy would improve if one used a separate model for each group instead of a single one for the whole dataset. However, there is no straightforward way of performing the grouping task, as series from disparate sources may look and behave similarly. Moreover, clustering the series using generic metrics may not be useful for improving the forecasting accuracy.

In this regard, the ensemble of specialists algorithm trains a number of models (NNs and per-series parameters) concurrently and forces them to specialize in a subset of series. The algorithm is summarized as follows:

1. Create a pool of models (e.g. seven models) and randomly allocate a part (e.g. half of the time series) to each model.
2. For each model:
 - (a) Execute a single training on the allocated subset.
 - (b) Record the performance for the whole training set (in-sample, average over all points of the training part of a series).
3. Rank the models for each series and then allocate each series to the top N (e.g. two) best models.
4. Repeat steps 2 and 3 until the average error in the validation area starts growing.

Thus, the final forecast for a particular series is the average of the forecasts produced by the top N models. The main assumption here is continuity: if a particular model is good at forecasting the in-sample part of the series, it will hopefully display accurate results in the out-of-sample part of the series as well. The architecture

and the inputs used for the individual models remain the same. What differs, and is manipulated actively, between epochs is the composition of the training data set for each model. Fig. 4 shows an example allocation of ten series among seven models altogether and two top models.

A simpler approach, called here simple ensemble, was used for monthly and quarterly data instead of the ensemble of specialists. In this case, the data were split into two non-overlapping sets at the beginning of each run and then models were trained and forecasts made for each of the two halves. This was a kind of bagging, and worked well too.

It is worth mentioning that the ensemble of specialists improved the forecasting accuracy by around 3% on the M3 monthly data set in the work of Smyl (2017). However, the difference reported was not stable, depending on the data and the quality of the models used. Therefore, more work is needed to delineate the areas of superiority of each method clearly.

3.2.3. Stage of training

The forecasts generated by a few (e.g. 4–5) of the most recent training epochs were ensembled to provide a single forecast. The whole training typically used 10–30 epochs; thus, in the case of 20 epochs for example, the final forecast was actually an average of the forecasts produced at epochs 16, 17, 18, 19 and 20.

3.3. Backtesting

Backtesting was implemented by removing typically one, but sometimes two, of the last horizon-number of points from each series (e.g. 18 or 36 for monthly data), and training the system on a set with such shortened series. However, while the training steps were never exposed to the removed values, the system was tested on the validation (removed) area after every epoch, and the results guided the architectural and hyperparameter choices. In practice, there was a very strong correlation

Table 1
Details of the architecture and parameters used.

Frequency	PF	PIs
Monthly	Simple ensemble Residual (1–3–6–12) LVP = 50 Epochs = 10 LR = $5e-4$ Max length = 272 Training percentile = 49 State size of LSTMs = 50	Epochs = 14 LR = $1e-3$, $\{8,3e-4\}$, $\{13,1e-4\}$ Max length = 512
Quarterly	Simple ensemble (1,2)–(4,8) LVP = 80 Epochs = 15 LR = $1e-3$, $\{10,1e-4\}$ Max length = 174 Training percentile = 45 State size of LSTMs = 40	Epochs = 16 LR = $1e-3$, $\{7,3e-4\}$, $\{11,1e-4\}$
Yearly	Ensemble of specialists 4/5 Attentive (1,6) LVP = 0 Epochs = 12 LR = $1e-4$, $\{15,1e-5\}$ Max length = 72 Training percentile = 50 State size of LSTMs = 30	Attentive (1,6),NL Epochs = 29 LR = $1e-4$, $\{17,3e-5\}$, $\{22,1e-5\}$
Daily	Ensemble of specialists 4/5 (1,3)–(7,14) Seasonality = 7 LVP = 100 Epochs = 13 LR = $3e-4$, $\{9,1e-4\}$ Max length = 112 Training percentile = 49 State size of LSTMs = 40	Epochs = 21 LR = $3e-4$, $\{13,1e-4\}$
Weekly	Ensemble of specialists 3/5 Attentive(1,52) Seasonality = 52 LVP = 100 Epochs = 23 LR = $1e-3$, $\{11,3e-4\}$, $\{17,1e-4\}$ Max length = 335 Training percentile = 47 State size of LSTMs = 40	Ensemble of specialists 4/5 Epochs = 31 LR = $1e-3$, $\{15,3e-4\}$
Hourly	Ensemble of specialists 4/5 (1,4)–(24,168) Seasonality = 24,168 LVP = 10 Epochs = 27 LR = $1e-2$, $\{7,5e-3\}$, $\{18,1e-3\}$, $\{22,3e-4\}$ Max length = NA Training percentile = 49 State size of LSTMs = 40	Epochs = 37 LR = $1e-2$, $\{20,1e-3\}$

between the validation results when testing on the last horizon-number of points and the penultimate horizon-number of points, with the former approach typically being used because it also admitted a larger number of series (many were too short to be used for backtesting if more than one horizon-number of points was removed).

While many series were short, there were also many that were very long, for example representing over 300 years of monthly data. The usefulness of the early parts of such series for the accuracy of the forecast was not obvious, while they involved an obvious computational demand. Thus, the long series were shortened, keeping only the most recent “maximum length” points. The

maximum length hyperparameter was tested and increased from a relatively small value until no further meaningful improvement in accuracy was observed in backtesting. It is listed in Table 1 along with the rest of the hyperparameters.

3.4. Hyperparameters

All hyperparameters were chosen using some combination of reasoning, intuition, and backtesting. The main tool used for preventing over-fitting was early stopping: during training, the average accuracy on the validation area (typically the last output horizon number of points,

see Fig. 1) was calculated after every training epoch. The epoch with the lowest validation error was noted and used as the maximum number of epochs when doing the final (using all of the data) learning and forecasting. The learning rate schedule was also decided by observing the validation errors after every epoch.

Table 1 lists all of the NN architectures and hyperparameters used. If a PI model used the same values as the PF model, the values are not repeated. A few comments about each:

- **Ensembling**
Either the simple ensemble or the ensemble of specialists. In the latter case it is detailed as `topN/numberOfAllModels`, e.g. 4/5, see Section 3.2.2. In the case of yearly data, both ensembling methods were tried, but in the cases of daily, weekly, and hourly data, the ensemble of specialists was chosen without experimentation, under the belief that it should provide better results.
- **NN architecture**
It is encoded as a sequence of blocks, in brackets, see Section 2.2.5. The residual shortcuts around the blocks, or, in the special case of LSTMs as per Kim et al. (2017), around the layers, are marked with dashes. Let me quickly describe the architecture of each type of series:
 - Monthly series used a single block of the special residual layers.
 - Quarterly, daily, and hourly series used what perhaps should be a standard architecture (as it appears to work well in other contexts, outside of the M4 Competition, too): two blocks of two dilated LSTM layers.
 - Point forecast models of yearly series used a single block of dilated LSTMs with attention, encoded as attentive (1,6), while models for prediction intervals added a standard dense layer with $\tanh()$ activation, which I call the nonlinear layer (NL), and therefore this is encoded as attentive (1,6), NL.
 - The architecture for weekly series, as described above, used attentive LSTMs.
 - As in other cases in the table, the architecture chosen was result of some reasoning/beliefs and experimentation. I believed that, in case of seasonal series, at least one of the dilations should be equal to the seasonality, while another should be in the range of the prediction horizon. It is likely that the architecture was over-fitted to the backtesting results; for example, the more standard architectures (1,3)–(6,12) or (1,3,12) would almost certainly work well for monthly series too (without the special residual architecture).
- **LVP**
LVP stands for level variability penalty, and is the multiplier that is applied to the level wiggleness penalty. It applies only to the seasonal models. The value is not very sensitive, as changing it even by 50% would not make a big difference. However, it is still important.

- **Number of epochs**

The number of training epochs for the final training and forecasting runs was chosen experimentally as the one that minimized the error on the validation area. There was a clear interplay between the learning rate and the number of epochs: higher learning rates needed smaller numbers of epochs. Another factor that influenced them both was the computational requirement of a subset: a larger number of series in a subset forced a preference for a smaller number of epochs (and thus higher learning rates).

- **Learning rates**

The first number is the initial learning rate, which was often reduced during training; for example, in case of the model for yearly PIs, it started at $1e-4$, but was reduced to $3e-5$ at epoch 17 and again to $1e-5$ at epoch 22. The schedule was result of observing the behaviors of the validation errors after each training epoch. When they plateaued for around two epochs, the learning rate was reduced by a factor of 3–10.

- **Max length**

This parameter lists maximum length of series used, see Section 3.3. In the case of hourly series, there was no chopping, all series were used in their original length.

- **Training percentile**

See Section 3.1.1.

- **State size of LSTMs**

LSTM cells maintain a vector of numbers, called the state, which is their memory. The size of the state was not a sensitive parameter, with values above 30 working well. Larger values slow down the calculations, but reduce the number of epochs needed slightly. There was no benefit in accuracy of using larger states.

3.5. Implementation

The method was implemented through four programs: two using the ensemble of specialists and two using simple ensembling, as was described earlier. Each pair consisted of one program for generating the PFs and another for estimating the PIs. If the competition were to happen today, probably only two programs would be needed, one using the ensemble of specialists and another using the simple ensemble, as PIs and PFs can be generated from a single program by modifying the loss function and the architecture. The method was written in C++ relying on the DyNet library (Neubig et al., 2017). It can be compiled and run on Windows, Linux or Mac, and can optionally write to a relational database, such as SQL Server or MySQL, to facilitate the analysis of the backtesting results, which is very useful in practice. The programs use CPU, not GPU, and are meant to be run in parallel. The code is available publicly at the M4 GitHub repository (<https://github.com/M4Competition/M4-methods>) to facilitate replicability and support future research (Makridakis, Assimakopoulos, & Spiliotis, 2018). The code is well commented and is the ultimate description of the method.

3.6. What did not work well and recent changes

The method generated accurate forecasts for most of the frequencies, but especially the monthly, yearly and quarterly ones. However, the accuracy was sub-optimal for the cases of the daily and weekly data. This can be explained in part by the author's concentration on the "three big" subsets: monthly, yearly and quarterly, as they covered 95% of the data, and performing well on them was key to success in the competition. However, subsequent work on daily and weekly data confirmed that under-performance on these frequencies is a real problem.

Since the competition ended, several improvements have been attempted. One such attempt that achieved noticeable improvements in accuracy on the daily and weekly data, bringing the performance to the level of the best benchmarks, is as follows. When analyzing the values of the smoothing coefficients, as they changed with passing training epochs, it became clear that they did not seem to plateau in late epochs, as the gradient descent did not seem to push them strongly enough. Thus, a separate, larger learning rate, which was a multiple of three of the main learning rate, was assigned to them, and this had the required effect. The smoothing coefficients changed quickly and eventually plateaued in late epochs.

4. Hybrid, hierarchical, and understandable ML models

This section begins by summarizing the main features of the model, then outlines generalizations and broader implications of its techniques and approaches. Also in this context, I retrace the steps that lead to the formulation of the models described in this paper.

The winning solution was a hybrid forecasting method which mixed exponential smoothing-inspired formulas, used for deseasonalizing and normalizing the series, with advanced neural networks, exploited for extrapolating the series. Equally important was the hierarchical structure of the method, which combined a global part learned across many time series (weights of the NN) with a time series specific part (smoothing coefficients and initial seasonality components). The third main component of the method was a broad usage of ensembling, at multiple levels. The first two features were made possible by the great functionalities offered by the modern NN systems of automatic differentiation and dynamic computational graphs (Paszke et al., 2017).

Automatic differentiation allows the building of models that utilize expressions made up of two sets: a quite broad list of basic functions, like $\sin()$, $\exp()$, etc., and a list of operators like matrix and element-wise multiplications, additions, reciprocal, etc. Neural networks that use matrix operations and some nonlinear functions are just examples of the allowed expressions. The gradient descent machinery fits parameters of all of these expressions. In the model described here, there was both a NN and a non-NN part (exponential smoothing inspired formulas). It is quite feasible to build models that encode complicated technical or business knowledge.

Dynamic computational graphs allow the building of hierarchical models, with global and local (here, per time

series) expressions and parameters. There could also be per-group parts. The models can be quite general; e.g. in a classical, statistical vein:

$$\text{Student performance} = \text{School impact} + \text{Teacher impact} + \text{Individual impact}.$$

Note that each component can be a separate NN, an inscrutable black box. However, we can observe and quantify the impact of each of the black boxes, both generally and in each case, and therefore we are getting a partially understandable ML model.

Automatic differentiation is also a fundamental feature of Stan, a probabilistic programming language (Carpenter, Hoffman, Brubaker, Lee, Li, & Betancourt, 2015). It fits models primarily using Hamiltonian Markov chain Monte Carlo, so the optimization is different, but the underlying auto-differentiation feels very similar. This similarity in modeling capabilities between Stan and DyNet led to the formulation of the proposed model, as is described in more detail below.

By the middle of 2016, I and my collaborators had successfully created extensions and generalizations of the Holt and Holt-Winters models in Stan (I called this family of models LGT: local and global trend models; see Smyl & Zhang, 2015, and Smyl, Bergmeir, Wibowo, & Ng, 2019), and experimented with using them along with NN models (Smyl & Kuber, 2016). Later, I also experimented a lot with building NN models for the M3 Competition data set (Smyl, 2017). I was able to beat classical statistical algorithms on the yearly (and therefore non-seasonal) subset, but could not do it on the monthly subset. For seasonal series, I used STL decomposition as part of the preprocessing, so clearly it did not work well. Also, my LGT models were more accurate than my NN models in every category of the M3 data. Thus, when I realized that DyNet, like Stan, allows a broad range of models to be coded freely, I decided to apply LGT ideas, such as dealing with seasonality, to a NN model. That is how the M4 winning solution was born.

Acknowledgments

I would like to thank Professor Spyros Makridakis and his colleagues for organizing the M4 Competition. I believe that forecasting competitions have been the main driver for the advancement of deep learning during its early years. Competitions enable the comparison of various forecasting methods, the exchange of ideas and the sharing of code. In addition, the competition dataset often becomes a valuable resource for years to come.

Finally, I would like to thank Evangelos Spiliotis for his help in editing this paper.

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. URL <https://www.tensorflow.org/>, software available from tensorflow.org.
- Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., & Betancourt, M. (2015). The stan math library: Reverse-mode automatic differentiation in c++. CoRR, abs/1509.07164.

- Chan, F., & Pauwels, L. L. (2018). Some theoretical results on forecast combinations. *International Journal of Forecasting*, 34(1), 64–74.
- Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., et al. (2017). Dilated recurrent neural networks. arXiv e-prints, arXiv:1710.02224.
- Dimoukas, I., Mazidi, P., & Herre, L. (2019). Neural networks for GEFCom2017 probabilistic load forecasting. *International Journal of Forecasting*, (in press).
- Gardner, E. S. (2006). Exponential smoothing: The state of the art – Part II. *International Journal of Forecasting*, 22(4), 637–666.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. arXiv e-prints, arXiv:1512.03385.
- Hyndman, R. J., Koehler, A. B., Ord, A. B., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer Verlag.
- Hyndman, R., Wang, E., & Laptev, N. (2015). Large-scale unusual time series detection. In *IEEE international conference on data mining*.
- Kang, Y., Hyndman, R. J., & Smith-Miles, K. (2017). Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, 33(2), 345–358.
- Kim, J., El-Khamy, M., & Lee, J. (2017). Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. arXiv e-prints, arXiv:1701.03360.
- Makridakis, S. (2017). The forthcoming artificial intelligence (AI) revolution: Its impact on society and firms. *Futures*, 90, 46–60.
- Makridakis, S., Assimakopoulos, V., & Spiliotis, E. (2018). Objectivity, reproducibility and replicability in forecasting research. *International Journal of Forecasting*, 34(4), 835–838.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018a). The M4 Competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4), 802–808.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS One*, 13(3), 1–26.
- Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastopoulos, A., et al. (2017). DyNet: The dynamic neural network toolkit. arXiv preprint, arXiv:1701.03980.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). Automatic differentiation in PyTorch. In NIPS 2017 autodiff workshop.
- Petropoulos, F., Hyndman, R. J., & Bergmeir, C. (2018). Exploring the sources of uncertainty: Why does bagging for time series forecasting work? *European Journal of Operational Research*, 268(2), 545–554.
- Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., & Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. arXiv e-prints, arXiv:1704.02971.
- Ramsay, J., & Silverman, B. (2002). *Functional data analysis*. New York: Springer-Verlag.
- Smyl, S. (2017). Ensemble of specialized neural networks for time series forecasting. In *37th international symposium on forecasting*.
- Smyl, S., Bergmeir, C., Wibowo, E., & Ng, T. W. (2019). Rlgt: Bayesian exponential smoothing models with trend modifications. R package version 01-2.
- Smyl, S., & Kuber, K. (2016). Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks. In *36th international symposium on forecasting*.
- Smyl, S., & Zhang, Q. (2015). Fitting and extending exponential smoothing models with Stan. In *35th international symposium on forecasting*.
- Takeuchi, I., Le, Q. V., Sears, T. D., & Smola, A. J. (2006). Nonparametric quantile estimation. *Journal of Machine Learning Research (JMLR)*, 7, 1231–1264.
- Taylor, J. W. (2003). Short-term electricity demand forecasting using double seasonal exponential smoothing. *The Journal of the Operational Research Society*, 54(8), 799–805.
- Weron, R. (2014). Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, 30(4), 1030–1081.



Machine learning methods for GEFCom2017 probabilistic load forecasting

Slawek Smyl^{a,*}, N. Grace Hua^b

^a Uber Technologies, USA

^b Microsoft, USA

ARTICLE INFO

Keywords:

Global energy forecasting competition
Quantile random forest
Gradient boosting
Neural networks
Deep learning
Ensemble forecasting
Probabilistic forecasting

ABSTRACT

This paper describes the preprocessing and forecasting methods used by team Orbuculum during the qualifying match of the Global Energy Forecasting Competition 2017 (GEFCom2017). Tree-based algorithms (gradient boosting and quantile random forest) and neural networks made up an ensemble. The ensemble prediction quantiles were obtained by a simple averaging of the ensemble members' prediction quantiles. The result shows a robust performance according to the pinball loss metric, with the ensemble model achieving third place in the qualifying match of the competition.

© 2019 Published by Elsevier B.V. on behalf of International Institute of Forecasters.

1. Introduction

This paper presents the approaches used by team Orbuculum for probabilistic hierarchical load forecasting in the GEFCom2017 competition. Both authors have substantial forecasting experience in the technology sector, but lack domain knowledge in the energy sector. In the qualifying match we were classified at third position, and in the final match, where this knowledge would have been even more helpful, at fourth position.

The methods applied by our team in the two stages were similar. In the qualifying match we preprocessed the input data, then used an ensemble of tree-based algorithms (quantile random forest, gradient boosted trees) and neural networks to generate the forecast with a pinball loss function, but in the final match we used an ensemble of tree-based algorithms and a naïve quantile approach. The reason for the exclusion of neural networks from the final match was their lack of stable backtesting performance. Thus, this paper concentrates on the first GEFCom2017 stage, since our methodologies were a bit richer during the qualifying match.

The rest of this paper is structured as follows. Section 2 describes our cross-validation framework. Two versions of preprocessing were used, and Section 3 examines them both in detail. Section 4 describes the models used and Section 5 explains how they were ensembled. Section 6 concludes the paper and discusses further research directions.

2. Cross-validation framework

The cross-validation framework was designed to follow the competition task closely, and was used for two purposes: (1) to tune the hyper-parameters and select features for individual algorithms; and (2) to compare the preprocessing and forecast backtesting performances of tuned models between the two authors, as well as to decide on ensemble approaches.

2.1. Input data

The data from the qualifying match, which was released in a rolling fashion, has been described by Hong, Xie, and Black (2019), so we will not go into detail here. However, we would like to point out one rather unusual feature of it: there was always a 1.5- to 2-month gap

* Corresponding author.

E-mail addresses: slaweks@hotmail.co.uk (S. Smyl), gracehua@gmail.com (N.G. Hua).

between the last known data and the beginning of the forecasting period. This meant that we had to give up the idea of using some features derived from the very recent period, and also forced us to use a similar gap (we used a gap of nine weeks) in our training and validation work.

2.2. Cross-validation folds

The competition required forecasts for four months in winter and early spring (January to April 2017). Thus, we do not need to test heavily for the rest of the months. As a result, we varied the backtesting forecast start date for each test year by rolling eight days, starting from January 1st. This gave us 12 testing folds in each test year. The choice of eight days is largely arbitrary, but the number chosen needs to be robust to weekly seasonality and to give a reasonable number of folds in each test year.

Also, since the input data date goes back to 2003, we had sufficient data to work with. We chose the years 2009 to 2016 as test years. All in all, we had 96 folds for each region. Note that we used all of the data: the data that were earlier than the nine weeks (the gap) before the beginning of the each backtesting forecast window were used for training.

3. Data preprocessing

The input data display seasonality at multiple frequencies, including daily (24 h), weekly (7 days), and yearly. We are not aware of any out-of-the-box time series R package that can be used to address these characteristics, and therefore decided to use supervised machine learning, after briefly exploring other customizable approaches such as the quantile regression spline, which showed a suboptimal performance.

Supervised machine learning algorithms learn to map inputs to outputs (note that both are usually of constant size). The algorithms themselves generally make no assumptions about the underlying data distribution (Breiman, 2001), and can be applied to regression or classification tasks. Careful data preprocessing needs to be done in order for them to be applied in the time series forecasting context, though the type of preprocessing applied depends on both the algorithm and the available data, including the numbers of variables (features) and samples.

The two authors use separate preprocessing approaches. Both approaches are described below and are illustrated in Fig. 1.

3.1. Data preprocessing stream 1

The electrical load time series have a pronounced yearly seasonality, caused primarily by changes in temperature and daylight length. Even the shape of the hourly patterns (7×24 per week) changes with the seasons. In such situations, it may be beneficial to reduce the learning period to the period of interest, so we preprocessed and learned only from a subset of the data, with the output window falling between the middle of December and the middle of May.

For most ML algorithms, which do not use a state, this reduction of the training data set is a reasonable choice without side effects. Unfortunately, though, it creates problems for algorithms with states, such as those with recurrent neural networks (RNN): large holes in the data would be disruptive and misleading. Thus, the decision to use the reduced data set had the side effect of rejecting the idea of using recurrent neural networks, which we would not consider necessary at present – while not learning from summer months, these months could have been used for keeping the RNN state updated. Alternatively, we could start to “walk through” a time series (not learning, just exposing the network to inputs) again some time before the training period begins, say at the beginning of each November.

We learned across all time series; in other words, we used the same model for learning and forecasting for all 10 zones.

We used the standard rolling window approach: input and output windows of a constant size were moved along each time series in tandem, extracting features and producing outputs at each step (for steps of one day).

3.1.1. Input window

This competition had a two-month gap between the last known data point and the beginning of the forecast window, so it was not possible to create features based on the recent load or temperature. However, we found through backtesting that longer-term features were still useful, and used the averages of the load recorded on the same day of the week and hour as in the matching output window (refer to the paragraph below on processing same day-of-the-week-and-hour values separately) in eight 4-week sub-windows. Thus, assuming that the hole was always nine weeks (erring on the safe side), the first sub-window started at $8 \times 4 + 9$ weeks before the beginning of the forecasting window, the next one started at $7 \times 4 + 9$ weeks before the beginning of the forecasting window, and so on.

The usefulness of the long-term features is interesting; perhaps the algorithms were able to pick up some longer-term dependencies of temperature among the summer, autumn and winter months.

3.1.2. Output window

The goal was to forecast hourly values of a whole month, a vector of over 700 values. Most of the ML algorithm implementations, including those we used, allow only a single scalar value as their output. While neural networks can output a vector, we considered this vector too large. Thus, some rearranging was needed.

We made the learning easier by partially “escaping seasonality”, i.e., processing values of the same day of the week and hour separately. Fig. 2 shows two (out of 168) short time series that were created in this way. Note that they are relatively flat. The learning can only deal with the effects of holidays and longer trends, not the strong intra-week fluctuations. The length is five, but the last point is not always used – these are points in the shaded area that belongs to the next month. Thus, the task of forecasting a single high frequency (hourly) series changes into the task

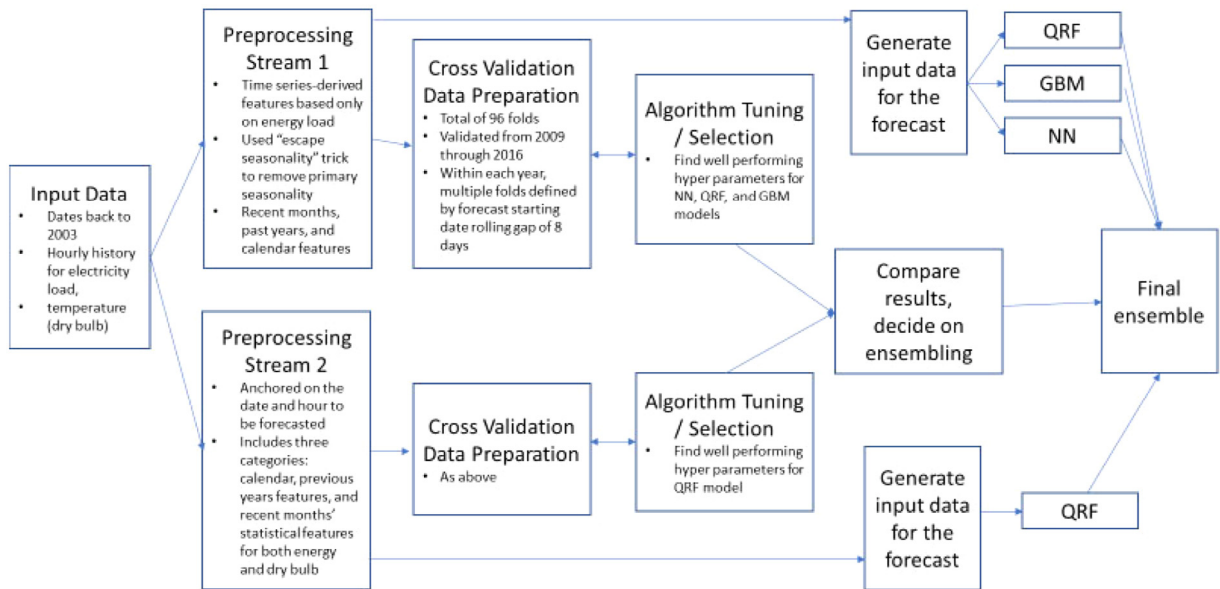


Fig. 1. Processes and the data flow.

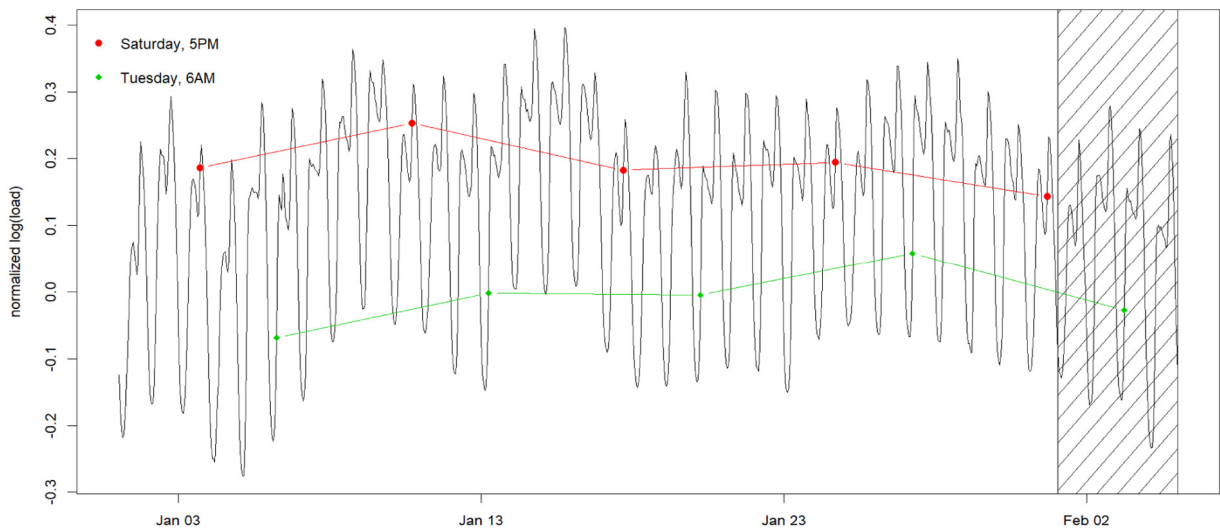


Fig. 2. Original hourly time series (black) and two examples of transformed series of the same day of the week and hour.

of forecasting 168 short, but much less variable, series of a lower frequency.

While a five-value vector is fine as an output for neural networks, it is still too large for the versions of boosted trees and random forests that we used, for which only scalars are possible. Thus, we made an additional rearrangement for these algorithms: for each original row with five outputs we created five preprocessed rows, each with one output value and almost the same input, except for one additional field that denoted the output position.

The results obtained using a QRF model working on data preprocessed in Stream 2, which did not use this 'escape seasonality' trick, suggest that it is not really necessary when a sufficiently large amount of data is available,

as was the case in this competition, allowing the algorithms can learn the seasonality thoroughly. However, our experience is that it is very useful when working with smaller data sets.

3.1.3. Normalization

Each model was using all series for learning. To enable it, it was necessary to normalize both the input time series features and the output values. Our preliminary analysis suggested that there was no noticeable trend in the energy consumption over the years. Thus, given the existence of the above-mentioned hole, which precludes the use of, for example, a median of the last few same-day-of-the-week-and-hour values as the normalization value, we decided to use a global average of a series as the

normalization value, referred to from now on as the *level*. This global characteristic of the *level*, while varying by series, meant that the “seasonality escape” was not thorough – the algorithms had to learn to forecast different days of the week and hours differently (the same hour-and-day-of-the-week series are relatively flat, but their amplitudes will vary).

We reduced the impact of outliers by using a $\log()$ transformation, so the normalization was $x_i \rightarrow \log(x_i/\text{level})$.

3.1.4. Calendar and holiday features

Even though the main 168-hours-per-week seasonality was partially escaped, we still added a full set of calendar features: hour (for NNs only, see below), day of the week, and week of the year (with week 52 being extended to cover the rest of the year).

Easter does not affect the energy consumption in the US in any noticeable way, and as this was the only moving holiday in the forecasting period, we did not make any special arrangements for Easter. However, we did create a feature that was equal to one on Christmas and New Year (and zero otherwise). Similarly, we created several features that were non-zero on a few days that, although not officially holidays, still had an impact on the energy consumption: Christmas Eve, Boxing Day, and January 2nd.

3.1.5. Other features

A few other features that we used included:

- the calendar year, normalized to run from zero to one;
- the quantile of the average load of a series within all loads of all series (all time series derived features are normalized, so this gives the algorithm the possibility of finding some difference in behavior between small- and large-value time series); and
- the id of a series (one-hot encoded vector of 10, so that the algorithm has a chance of learning the quirks of individual series).

Fig. 3 depicts an example backtesting step. The filled rectangle denotes a given forecasting window (output), 31 days from 15 January to 15 February 2016, while the crossed rectangles indicate the part of the preprocessed data, from 8 to 22 January of several previous years, that is used for training. It should be kept in mind that each of the preprocessed points in the crossed area (15×24 in each year) contains a relatively long vector of preprocessed inputs and five weekly separated outputs.

3.2. Data preprocessing stream 2

3.2.1. Additional source data for features

DryBulb is the air temperature. The temperature of the last few days can be useful for short-term forecasting (Wang, Liu, & Hong, 2016), and it is common practice to use temperature forecasts as an input to the load forecast, due to the strong correlation between the two in actual observations. However, given the big gap between the last observation in the data release and the first forecast

date (as described above), along with the medium-length forecast horizon (one month), we did not expect the dry bulb forecast to serve as a reliable input to the load forecasting task. However, this preprocessing stream made use of DryBulb past observations as input features, using the same transformations as were applied to the load; see below. No DewPnt data was used in any of our training.

In addition to standard calendar features, this preprocessing stream also used federal holidays, as defined by the US Office of Personnel Management (OPM).¹

3.2.2. Input and output structure

In this preprocessing stream, the output is treated as a column vector; i.e., in a 31-day month, the forecast is considered as a 744×1 vector. For each element of this vector, we engineer a feature vector for training the ML algorithms. No separation/rearrangement is done to break down this column vector.

Even though the competition only required forecasting for January to April, features of all other months were still included for training and testing. This is similar to statistical time series approaches, where, in general, for a seasonal time series, all data leading up to the forecast will be used to train the models even if the goal is only to generate forecasts for one specific season.

3.2.3. Special callouts

Despite the availability of abundant data (more than 10 years' worth), it is not always optimal to use all available history for training. In fact, the length of the training history became one of the hyperparameters that was tuned based on the cross-validation set.

ML approaches generally do not deal well with unnormalized data. Before engineering all statistical features as described below, both Load and DryBulb raw data were normalized by simply subtracting the minimum value and dividing by maximum. This brings all values to a consistent range. It's more common for the normalization to be divided by the difference between maximum and minimum. But in this context, maximum minus minimum is close enough to maximum for the calculation to be done using maximum alone.

3.2.4. Feature descriptions

There were three categories of features: calendar, previous years and recent months' features.

Calendar features had the following categories, and were engineered as follows:

(1) Federal holidays, one-hot encoded, based on the US office of Personnel Management's schedule;

(2) Year, month, day of week, day of month, hour of day, week of the year are one hot encoded. Also, a continuous feature was included for all of them. Take month as example, January's continuous feature is 1/12, February 2/12 and so on.

(3) Day of year, only continuous feature was engineered.

¹ <https://www.opm.gov/policy-data-oversight/snow-dismissal-procedures/federal-holidays>.

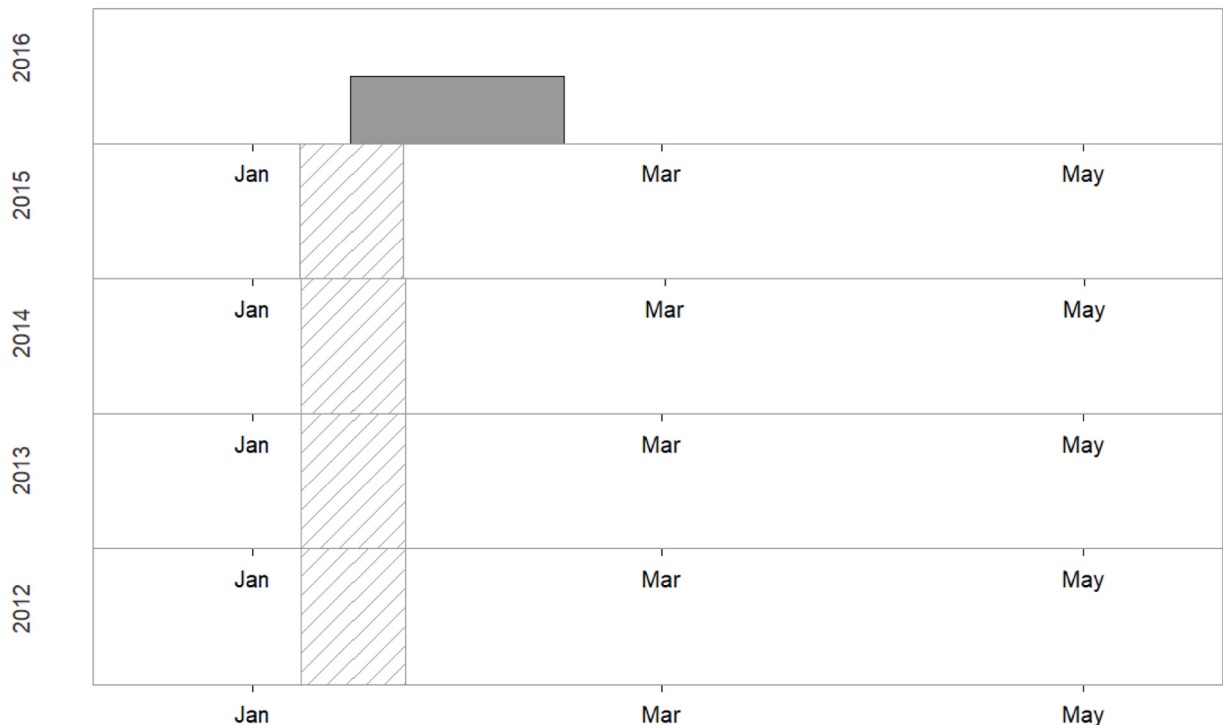


Fig. 3. An example step during backtesting. The solid color represents the forecasting period, while the hashed area is the part of the preprocessed data that is used for training.

Previous year's features serve for algorithms to learn "expected behaviors" that correspond to each seasons of the year. The same preprocessing approaches were applied to both previous year's normalized Load and Dry-Bulb data. The statistical features collected include: quantiles of 0.01, 0.05, 0.2, 0.5, 0.8, 0.95, 0.99, mean, and standard deviation. The procedure works as follows: For every single record, find all the records from all available previous years with the same day of the week, same hour of the day, week of the year, ranging from the week before to the week after; calculate the above statistics, and use as features.

It is obvious that the previous years' features alone will not be sufficient, as both the temperature and the energy consumption fluctuate from year to year. The features from recent months were engineered to reflect the recent trends. However, these features cannot be generated using the most recent past, due to the hole between last training date and first forecast date. Instead, a gap of at least 14 weeks was always used, to guarantee a period of at least three months between the last available training data and the last required forecast. The same procedures were applied to both the Load and DryBulb data as in the case of the previous years' features, and we collected quantiles of 0.01, 0.05, 0.2, 0.5, 0.8, 0.95 and 0.99, along with the mean and the standard deviation. The procedure worked as follows: for each record, find all records from 14–19 weeks prior to it, with the same day of the week, hour of the day and week of the year, then calculate the above statistics and use the results as features.

4. Models

We used the following algorithms/tools:

- Microsoft's Fast Forest,² which is a version of the quantile random forest (QRF; see Meinshausen, 2006).
- Gradient boosting machine (GBM; see Friedman, 2002) – an R package (Ridgeway, Edwards, and Krieglner).
- A multi-layer perceptron (MLP) with shortcuts, CNTK (Microsoft Cognitive Toolkit).

These were chosen because of both our familiarity with them and their ability to fit quantile regression models, and therefore to produce density forecasts directly.

4.1. GBM model

We used 24 independent models for each hour, partly due to performance issues with the GBM algorithm, but also following Gaillard, Gaude, and Nedellec (2016). The preprocessed data set covered the winter (competition) months, but backtesting and final training was conducted using even smaller chunks that included only 15 days of data, eight days before the first forecast day and seven days after. For example, when forecasting for January, the set would include all preprocessed records from 23rd December to 7th January of all preceding years, but not the current year because of the gap.

² <https://msdn.microsoft.com/en-us/library/azure/dn913093.aspx>.

Two important parameters that we used, which were established by backtesting, were: shrinkage = 0.005, number of iterations (trees) = 1000.

GBM can run in quantile regression mode, but each percentile (10, 20, ..., 90) requires a separate run, so we had nine models for each hour. We used the `doParallel` package of R to speed things up.

4.2. QRF model

The two processing streams were both used on the QRF model, and both yielded solid performances in cross-validation.

In the first stream, we used the following parameters that we established through backtesting: number of leaves = 50, minimum number of cases in a leaf = 1, number of trees = 150. When using the features in the second stream, the optimal set of hyperparameters turned out to be 80 trees and 40 leaves.

4.3. Neural network model

4.3.1. Deep learning or shallow learning

It is tempting to think that the success of deep learning (DL) in image analysis, speech recognition and language processing could be replicated easily in the area of time series forecasting, and as a consequence that the data preprocessing required when using neural networks could be almost none. However, such is not typically the case. The main issue may be the amount of data available; e.g. even small images have thousands of pixels and the datasets involved have thousands to millions of samples, while there tend to be far fewer time series to consider, and each has many fewer points.

The temporal aspect is important too: a series needs to cover a long enough period of time relative to the forecasting horizon, irrespective of how frequently it is sampled.

Thus, it is usually worthwhile to help a neural network a good preprocessing, and the neural network architectures applied to time series forecasting tend to be shallow, i.e., containing only a few layers.

Loss function

There are several ways of developing neural networks for executing quantile regression, and therefore for producing prediction intervals. Probably the simplest and most correct way is to use the pinball loss as the learning loss function. During the competition we used BrainScript configuration interface for CNTK; at present it is possible to code the whole program, including the loss function, using R or Python.

Obtaining prediction intervals in this way is precise, easy and almost assumption-free; however, as in the case of GBM, a single run can learn to predict only one quantile at a time, so we ran it nine times for each quantile requested, in three parallel streams on a single graphics card. Had there been a need for more quantiles, as in GEFCom 2014, where the task involved the forecasting of 99 percentiles, we would merely have interpolated between adjacent quantiles. Also, one must make sure that the quantiles obtained through different runs are

monotonic, so we checked them all, and sorted them if necessary.

Architecture and learning parameters

While our neural network was not really deep by current standards, it did use several layers with `tanh()` nonlinear transfer functions. It is very difficult to use more than two hidden layers when using the traditional feed-forward architecture, because of the “vanishing gradient” problem: the squashing functions make the gradient smaller and smaller during the backpropagation step (Hochreiter, Bengio, Frasconi, & Schmidhuber, 2001).

In classifications tasks such as image recognition, this problem has largely been solved by using a “linear rectifier”, or similar-shaped transfer function, instead of a squashing function like `tanh()` or a sigmoid. The rectifier is zero for negative input values and equal to the input elsewhere (Glorot, Bordes, & Bengio, 2011).

However, our experience suggests that the rectifier is not the best choice for a regression, as squashing functions offer the necessary nonlinearity and precision. Fortunately, another breakthrough in deep learning, namely the residual neural network (He, Zhang, Ren, & Sun, 2015), can also be applied to a sequence of layers with squashing functions. Fig. 4 shows the architecture used: several fully-connected layers, but with additional shortcuts or bypasses. These shortcuts add the original input to the layer output, thus converting the neural network into an additive model, where later layers add only a little to the output of the previous layer. The shortcuts make the gradient much less likely to vanish, meaning that many layers can be stacked, although we have found that there is seldom any benefit in going deeper than four layers for time series forecasting, and often two layers is sufficient.

The preprocessed input consisted of 92 values, while the output size was five (week-separated) values. We experimented with various batch sizes, from 10 to 256, and found the size to be not too important, with larger sizes improving the processing speed but slightly worsening the accuracy. We used the Adagrad optimizer, and injected Gaussian noise (to disturb the gradients) at a rate of 0.0005. We also applied L2 regularization on weights of 0.001. The initial learning rate for each sample was $1e-6$.

5. Ensemble

During the first five rounds, we ran three forecasting models: a NN, a GBM, and a QRF-based model. For the last submission, the ensemble was enlarged with one more QRF-based forecast, with its own different preprocessing. Our backtesting suggested that a simple average of the respective quantiles produced by the three or four models improved the accuracy over both the single best model (the QRF-based one) and an aggregation of the medians; thus, for example, the final forecast of the 90th percentile was an average of the 90th percentiles produced by each of the three or four models that constituted the ensemble.

Table 1 shows the metrics of the top performing algorithms and ensembles based on cross-validation. It is clear that the ensembles outperform even the best-performing individual algorithm.

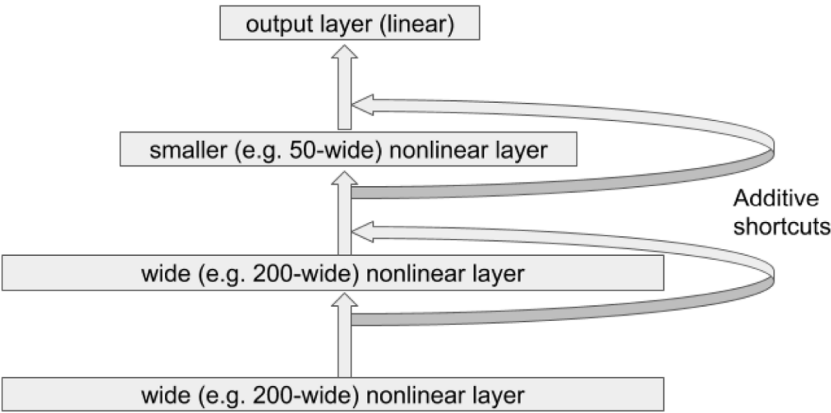


Fig. 4. Neural network architecture.

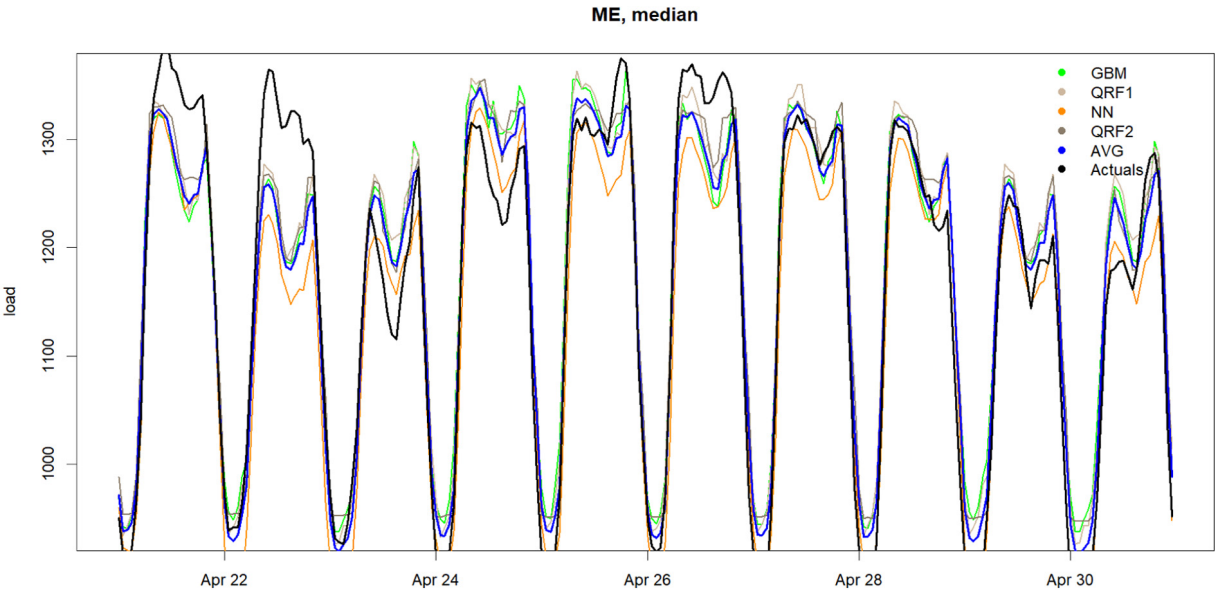


Fig. 5. An example forecast of the median for ME: our four algorithms, their average, and the actual.

Table 1
Bias, MAPE and pinball loss comparison of the top-performing algorithms.

predictionAlgorithm	pcBias	wMape	pbLoss
Avg of top 4	0.03	4.82	3.72
Avg of top 3	0.43	4.99	3.86
QRF + Preprocessing2	0.12	4.98	3.86
QRF + Preprocessing1	0.51	5.03	3.88
GBM + Preprocessing1	0.66	5.02	3.91
NN + Preprocessing1	−1.28	5.22	4.08

Fig. 5 may be helpful in this context: of the top four approaches, NN tends to under-forecast, as depicted by negative bias, while the other three models generally over-forecast. The ensemble effectively cancels this out and generates the forecast with the best cross-validated result. Of course, there are times when all four models are in agreement but wrong, as was the case during the first day in Fig. 5.

5.1. Hierarchical forecasting

There were 10 zones to forecast: all eight ISO New England zones, Massachusetts (the sum of the three zones under Massachusetts), and the total (the sum of the first eight zones), so this was partly a hierarchical forecasting problem. However, as we were producing density forecasts (10th, 20th...90th percentiles) and there was no requirement for consistency across the hierarchy (the quantiles did not need to add up), the contestants did not have to consider the hierarchical setup, though they could choose to use it to try to improve their forecasts. The reconciliation of density forecasts is an active research area (see for example the work of Taieb, Taylor, & Hyndman, 2017), and though we did not have the tools to attempt such a reconciliation during the competition, in future we would like to investigate the impact of such a reconciliation on the forecasts created using an ensemble of density forecasts.

6. Conclusion

We have presented the ways of preprocessing data and creating forecasts that we applied during GEFCom 2017. We achieved good results by employing rather standard preprocessing methods and standard or close-to-standard machine learning models. Also, it is worth noting that, while the data were not scarce, they were not rich either; i.e., all of the features used were derived either directly or indirectly (like holidays) from univariate time series (partly bivariate in preprocessing stream 2). In our experience, ML algorithms perform better when there are external, meaningful regressors.

Additional steps that would be worth taking in future include using further improvements in preprocessing, e.g. trying to learn why preprocessing stream 2 performed better as the input to the QRF (was it due mostly to the inclusion of the temperature-derived features?); perhaps replacing the non-recurrent NNs with recurrent ones; finding better ways of ensembling the individual density forecasts; and employing a hierarchical reconciliation of ensemble forecasts.

We would like to thank Dr. Tao Hong and everyone else who helped in organizing the competition. We second several of the suggestions of Breiman (2001), including those regarding the usefulness of the algorithmic approaches and the competitions for assisting in the progress of modeling and forecasting.

References

- Breiman, L. (2001). Statistical modeling: the two cultures (with comments and a rejoinder by the author). *Statistical Science*, 3, 199–231.
- Friedman, J. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38, 367–378.
- Gaillard, P., Gaude, Y., & Nedellec, R. (2016). Additive models and robust aggregation for GEFCom2014 probabilistic electric load and electricity price forecasting. *International Journal of Forecasting*, 32, 1038–1050.
- Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the 14th international conference on artificial intelligence and statistics* (pp. 315–323).
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. ArXiv preprint [arXiv:1512.03385](https://arxiv.org/abs/1512.03385).
- Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer, & J. F. Kolen (Eds.), *A field guide to dynamical recurrent neural networks*. IEEE Press.
- Hong, T., Xie, J., & Black, J. (2019). Global energy forecasting competition 2017: hierarchical probabilistic load forecasting. *International Journal of Forecasting*, 35(4), 1389–1399.
- Meinshausen, N. (2006). Quantile regression forests. *Journal of Machine Learning Research (JMLR)*, 7, 983–999.
- Ridgeway, G., Edwards, D., & Kriegler, B. (2013). Package 'gbm', version 2.1.
- Taieb, S. B., Taylor, J. W., & Hyndman, R. (2017). Coherent probabilistic forecasts for hierarchical time series. In *Proceedings of the 34th international conference on machine learning*.
- Wang, P., Liu, B., & Hong, T. (2016). Electric load forecasting with recency effect: A big data approach. *International Journal of Forecasting*, 32, 585–597.

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

Grzegorz Dudek[✉], Paweł Pełka[✉], and Sławek Smył[✉]

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).

Sławek Smył 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 MTLF 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 in [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 par 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

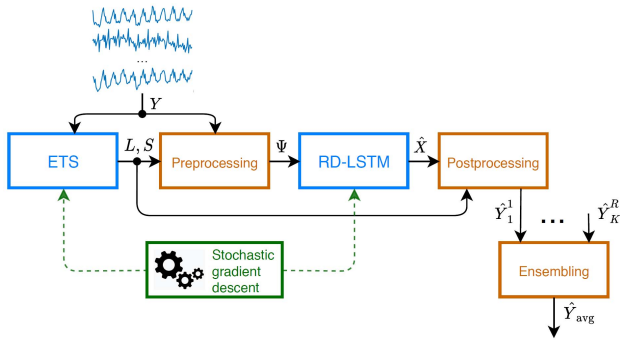


Fig. 1. Block diagram of the ETS+RD-LSTM forecasting system.

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]:

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

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 $\alpha, \beta \in [0, 1]$ 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

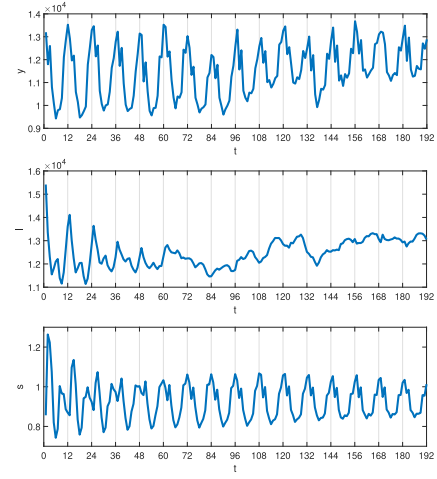


Fig. 2. Original time series and its level and seasonal components.

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 $\mathbf{x}_t^{in}$). 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 $\mathbf{x}_t^{out}$). 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, $\log(\cdot)$, is applied. The resulting preprocessing can be expressed as follows:

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

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

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

- 2) Second pair of input and output windows

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

- 3) ...

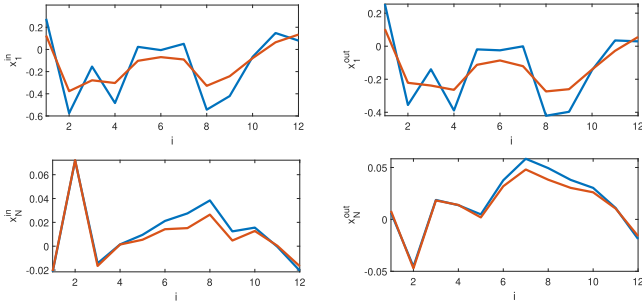


Fig. 3. Examples of the input (left) and output (right) vectors created for the time series shown in Fig. 2. Top: x -vectors for the first two cycles. Bottom: x -vectors for the last two cycles. The x -vectors created in the first training epoch in blue, in the last epoch in red.

4) N th pair of input and output windows:

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

These vectors are included in the training subset for the i th time series: $\Phi_i = \{(\mathbf{x}_t^{\text{in}}, \mathbf{x}_t^{\text{out}}) : t = 1, 2, \dots, N\}$. The training subsets for all M time series are combined and form the training set $\Psi = \{\Phi_1, \Phi_2, \dots, \Phi_M\}$, 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:

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

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

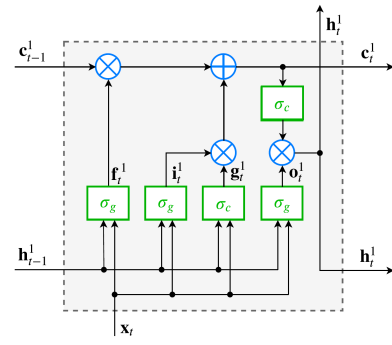


Fig. 4. LSTM block.

$\mathbf{c}_t$ 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].

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

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

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

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

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

Formulas related to the gates are as follows:

$$\mathbf{f}_t^1 = \sigma_g(\mathbf{W}_f^1 \mathbf{x}_t + \mathbf{V}_f^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_f^1) \quad (6)$$

$$\mathbf{i}_t^1 = \sigma_g(\mathbf{W}_i^1 \mathbf{x}_t + \mathbf{V}_i^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_i^1) \quad (7)$$

$$\mathbf{g}_t^1 = \sigma_c(\mathbf{W}_g^1 \mathbf{x}_t + \mathbf{V}_g^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_g^1) \quad (8)$$

$$\mathbf{o}_t^1 = \sigma_g(\mathbf{W}_o^1 \mathbf{x}_t + \mathbf{V}_o^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_o^1) \quad (9)$$

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

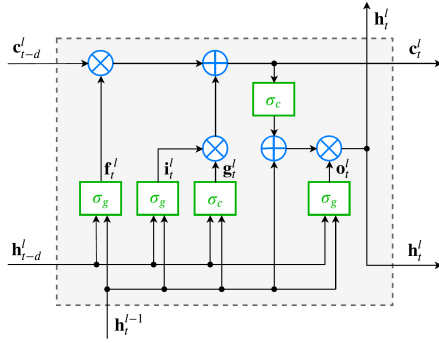


Fig. 5. RD-LSTM block.

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, $\mathbf{h}_{t-1}^{l-1}$ 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 $\mathbf{h}_{t-1}^{l-1}$ and transformed cell state $\sigma_c(\mathbf{c}_t^l)$. These transformations are not necessary because, in our case, the dimensions of $\mathbf{h}_{t-1}^{l-1}$ 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:

$$\mathbf{c}_t^l = \mathbf{f}_t^l \otimes \mathbf{c}_{t-d}^l + \mathbf{i}_t^l \otimes \mathbf{g}_t^l \quad (10)$$

$$\mathbf{h}_t^l = \mathbf{o}_t^l \otimes (\sigma_c(\mathbf{c}_t^l) + \mathbf{h}_{t-1}^{l-1}) \quad (11)$$

$$\mathbf{f}_t^l = \sigma_g(\mathbf{W}_f^l \mathbf{h}_{t-1}^{l-1} + \mathbf{V}_f^l \mathbf{h}_{t-d}^l + \mathbf{b}_f^l) \quad (12)$$

$$\mathbf{i}_t^l = \sigma_g(\mathbf{W}_i^l \mathbf{h}_{t-1}^{l-1} + \mathbf{V}_i^l \mathbf{h}_{t-d}^l + \mathbf{b}_i^l) \quad (13)$$

$$\mathbf{g}_t^l = \sigma_c(\mathbf{W}_g^l \mathbf{h}_{t-1}^{l-1} + \mathbf{V}_g^l \mathbf{h}_{t-d}^l + \mathbf{b}_g^l) \quad (14)$$

$$\mathbf{o}_t^l = \sigma_g(\mathbf{W}_o^l \mathbf{h}_{t-1}^{l-1} + \mathbf{V}_o^l \mathbf{h}_{t-d}^l + \mathbf{b}_o^l) \quad (15)$$

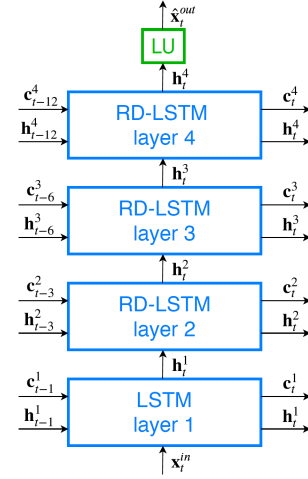


Fig. 6. RD-LSTM network architecture.

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

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

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 $t = 1, 2, \dots$. 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 t th learning cycle, RD-LSTM modifies its parameters depending on the forecast error for the t th 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 t th 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, $\Psi_1, \Psi_2, \dots, \Psi_K$, are averaged. The training set, $\Psi = \{\Phi_1, \Phi_2, \dots, \Phi_M\}$, is composed of subsets Φ_i containing the training samples for the i th time series. To create the training Ψ -subsets, first, a set of M time series is split randomly into K subsets of similar size: $\Theta_1, \Theta_2, \dots, \Theta_K$. The k th Ψ -subset contains the Φ -subsets for all time series excluding those in Θ_k , i.e., $\Psi_k = \Psi \setminus \{\Phi_i\}_{i \in \Theta_k}$. 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.

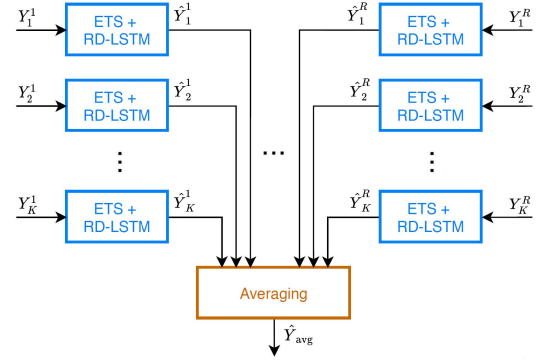


Fig. 7. Ensembling.

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 k th Ψ -subsets in the r th 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

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

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 k th one in the r th 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

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

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

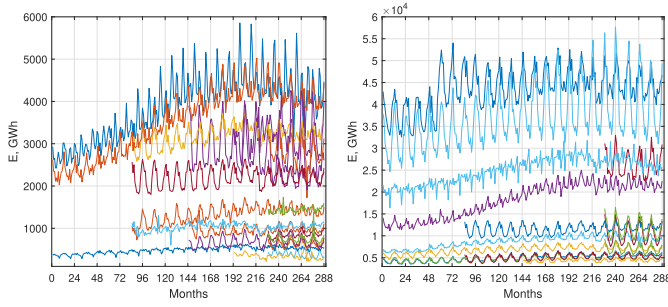


Fig. 8. Monthly electricity demand time series for 35 European countries.

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 wiggleness 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

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

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

The level wiggleness 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 `anfisc` 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].

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

- 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.

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

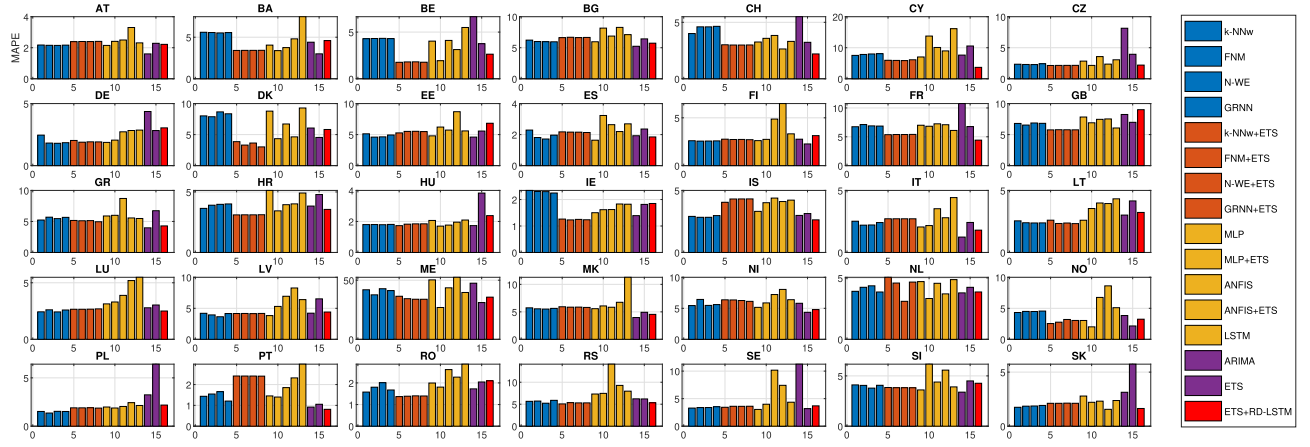


Fig. 9. MAPE for each country.

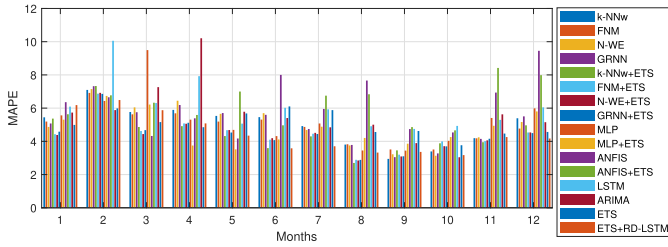


Fig. 10. MAPE for each month of the forecast period.

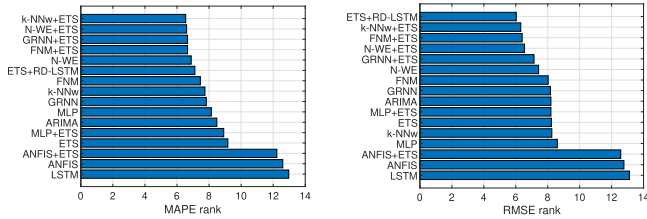


Fig. 11. Rankings of the models.

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 $\text{MAPE} = 2\%$, 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

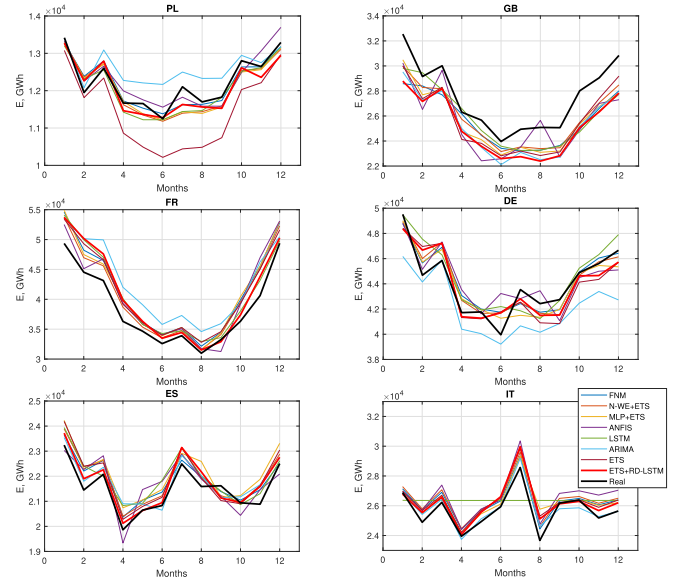


Fig. 12. Real and forecast monthly electricity demand for selected countries.

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 $\text{MAPE} = 8.52\%$ was the least accurate model, and for FR data, with $\text{MAPE} = 5.49\%$, 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

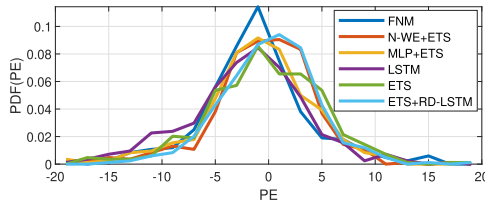


Fig. 13. PDF of the PEs.

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.

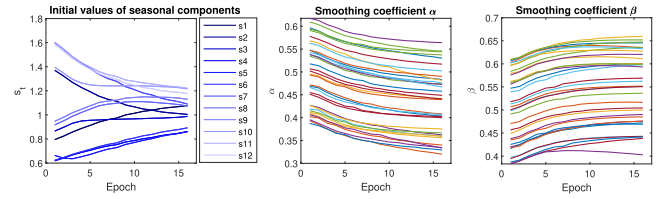


Fig. 14. ETS parameters during training.

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 winning 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 loses 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 transferring 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. Peřka 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. Pelka, "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. Carpow, 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. Pelka 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. Pelka 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. Tiño, "Managing diversity in regression ensembles," *J. Mach. Learn. Res.*, vol. 6, pp. 1621–1650, Dec. 2005.
- [44] N. Passalis, A. Tefas, J. Kannianen, 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 Pelka 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.



Sławek 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.



3ETS+RD-LSTM: A New Hybrid Model for Electrical Energy Consumption Forecasting

Grzegorz Dudek¹✉ , Paweł Pełka¹ , and Sławek Smył²

¹ Electrical Engineering Faculty, Częstochowa University of Technology, Częstochowa, Poland

{dudek,p.pelka}@el.pcz.czest.pl

² Uber Technologies, 555 Market Street, San Francisco, CA 94104, USA
slaweks@hotmail.co.uk

Abstract. This work presents an extended hybrid and hierarchical deep learning model for electrical energy consumption forecasting. The model combines initial time series (TS) decomposition, exponential smoothing (ETS) for forecasting trend and dispersion components, ETS for deseasonalization, advanced long short-term memory (LSTM), and ensembling. Multi-layer 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. Deseasonalization and LSTM are combined in a simultaneous learning process using stochastic gradient descent (SGD) which leads to learning TS representations and mapping at the same time. To deal with a forecast bias, an asymmetric pinball loss function was applied. Three-level ensembling provides a powerful regularization reducing the model variance. A simulation study performed on the monthly electricity demand TS for 35 European countries demonstrates a high performance of the proposed model. It generates more accurate forecasts than its predecessor (ETS+RD-LSTM [1]), statistical models such as ARIMA and ETS as well as state-of-the-art models based on machine learning (ML).

Keywords: Exponential smoothing · Long short-term memory · Mid-term load forecasting

1 Introduction

The power system load is a nonlinear and nonstationary process that can change rapidly due to many factors such as macroeconomic variations, weather, electricity prices, consumer types and habits, etc. Therefore, electricity demand forecasting, which is essential for the power system operation and planning, is a big

The project financed under the program of the Polish Minister of Science and Higher Education titled “Regional Initiative of Excellence”, 2019–2022. Project no. 020/RID/2018/19, the amount of financing 12,000,000.00 PLN.

© Springer Nature Switzerland AG 2020

H. Yang et al. (Eds.): ICONIP 2020, LNCS 12534, pp. 519–531, 2020.

https://doi.org/10.1007/978-3-030-63836-8_43

challenge. In this study we consider mid-term electrical load forecasting (MTLF) focusing on monthly electricity demand forecasting over 12 months horizon.

MTLF methods can be roughly classified into statistical/econometrics methods or ML methods [2]. The former include ARIMA, ETS and linear regression (LR). ARIMA and ETS can deal with seasonal TS but LR requires additional operations such as decomposition or extension of the model with periodic components [3]. Limited adaptability of the statistical MTLF models and problems with nonlinear relationship modeling have increased researchers' interest in ML and AI tools [4]. Of these, neural networks (NNs) are the most popular because of their attractive features including learning capabilities, universal approximation property, nonlinear modeling and massive parallelism. Some examples of using NNs for MTLF are: [5] where NN uses historical loads and weather variables to predict monthly demand and is trained by heuristic algorithms to improve performance, [6] where Kohonen NN is used, [7] where NNs are supported by fuzzy logic, [8] where generalized regression NN is used, [9] where weighted evolving fuzzy NN is used, and [10] where NNs, LR and AdaBoost are combined.

Recent trends in ML such as deep recurrent NNs (RNNs), are very attractive for TS forecasting [11]. RNNs are able to exhibit temporal dynamic behavior using their internal state to process sequences of inputs. Recent works have reported that RNNs, such as the LSTM, provide high accuracy in forecasting and outperform most of the traditional statistical and ML methods [12]. Some application examples of LSTMs to load forecasting can be found in [13–15].

In [1] we proposed a hybrid residual dilated LSTM and ETS model (ETS+RD-LSTM) for MTLF. This model was based on the winning submission to the M4 forecasting competition 2018 [16], developed by Slawek Smyl [17]. A simulation study confirmed the high performance of the model and its competitiveness with classical models such as ARIMA and ETS as well as state-of-the-art ML models. In this work we extend ETS+RD-LSTM by introducing initial TS normalization, i.e. detrending and unifying the variance. This method of TS preprocessing we used in our previous works achieving very good results [18]. Recently we used it for LSTM model obtaining a 15% reduction in error [19]. We expect that the TS initial normalization, which simplifies the relationship between input and output data, allows ETS+RD-LSTM to improve its performance.

2 Forecasting Model

The proposed model is a modified version of ETS+RD-LSTM which we described in [1]. We extend ETS+RD-LSTM by introducing initial TS normalization. A normalization procedure removes a trend and unifies variance of the TS. The normalized TS exhibit yearly patterns which are further removed using deseasonalization as an integral part of ETS+RD-LSTM. The normalized and deseasonalized TS are forecasted using RD-LSTM. Then the forecasts are reseasonalized using seasonal components extracted by ETS. To reduce the model variance, we use ensembling at three levels which aggregates individual forecasts.

The resulting aggregated forecasts are finally denormalized. For denormalization, the forecasts of the mean yearly demand and yearly dispersion are needed. They are produced by additional two ETS modules.

2.1 Architecture and Features

An architecture of the proposed forecasting system is shown in Fig. 1. The system components are as follows (symbols in *italics> denote the sets of TS or forecasts):*

- Normalization – each original monthly electricity demand TS is normalized. This procedure removes a trend from the TS and unifies its variance. Normalization module loads a set of TS (Z), calculates the series of yearly mean demands ($\bar{Z}$) and yearly dispersions (Σ) for each TS, and determines normalized series (Y).
- ETS – exponential smoothing modules for forecasting the yearly mean demands and their dispersions. These values are necessary for denormalization.
- Deseasonalization – each normalized TS is deseasonalized. This procedure extracts the seasonal components, S , individually for each series using ETS (ETSd module), and determines deseasonalized TS, X .
- RD-LSTM – residual dilated LSTM for forecasting the normalized and deseasonalized TS, X .
- Reseasonalization – each TS forecast produced by RD-LSTM is reseasonalized using inverse operations to deseasonalization.
- Ensembling – the reseasonalized forecasts produced by RD-LSTM are averaged. The ensembling module receives the sets of individual forecasts, $\hat{Y}_k^r$, and returns an aggregated forecast for each TS, $\hat{Y}_{\text{avg}}$.
- Denormalization – the averaged forecasts $\hat{Y}_{\text{avg}}$ are denormalized using forecasted values of the yearly means, $\hat{\bar{Z}}$, and dispersions, $\hat{\Sigma}$.

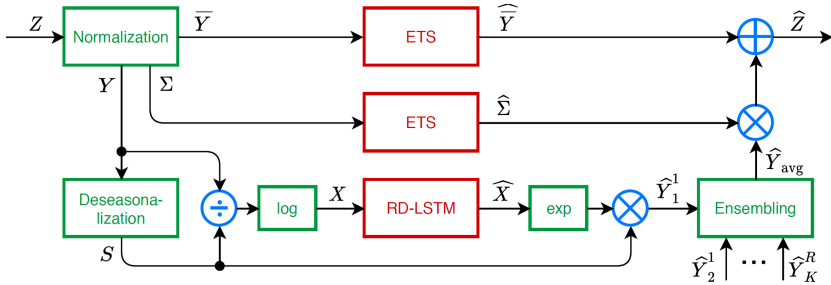


Fig. 1. The proposed forecasting system architecture

The proposed system has a hybrid and hierarchical structure. It combines statistical modeling (ETS), advanced ML (RD-LSTM), and ensembling. ETS is

used as a forecasting model for yearly means, $\bar{Z}$, and dispersions, Σ , as well as for extraction of seasonal components (ETSd). The preprocessed TS, without trend and seasonal variations, are forecasted using RD-LSTM. Details of data preprocessing and flow are described in Subsect. 2.2.

The TS are exploited in a hierarchical manner, meaning that both local and global components are utilized in order to extract and combine information at either a series or a dataset level, thus enhancing the forecasting accuracy. The global features are learned by RD-LSTM across many TS (cross-learning). The specific features of each individual TS, such as trend, variance, and seasonality, are extracted by normalization and ETSd modules. Thus, each series has a partially unique and partially shared model.

The strength of RD-LSTM, which revealed in M4 competition, is cross-learning, i.e., using many series to train a single model. This is unlike standard statistical TS algorithms, where a separate model is developed for each series. Another important ingredient in the success of the proposed method precursor in the M4 competition was the on-the-fly preprocessing that was an inherent part of the training process. Crucially, the parameters of this preprocessing (in the proposed model these are twelve initial seasonal components and smoothing coefficient β , see Subsect. 2.2) were being updated by the same overall optimization procedure (SGD) as weights of RD-LSTM, with the overarching goal of minimizing forecasting errors. This enables the model to simultaneous optimization of data representation, i.e. searching for the most suitable representations of input and output data for RD-LSTM, and forecasting performance.

ETSd is used as the preprocessing tool. It extracts a seasonal component which is used for deseasonalization of the normalized TS. ETSd was inspired by the Holt-Winters multiplicative seasonal model. However, it has been simplified by removing trend and level components (see Subsect. 2.2). This is because the input TS are normalized, i.e. they have no trend and their level is one. ETSd is optimized simultaneously with RD-LSTM using pinball loss function [17]:

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

where x_t and $\hat{x}_t$ are the actual and forecasted values, respectively, and $\tau \in (0, 1)$ is a parameter controlling the loss function asymmetry.

When $\tau = 0.5$ the loss function is symmetrical and penalizes positive and negative deviations equally. When the model 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 loss function, penalizing positive and negative deviations differently, allows the method to deal with bias.

ETS for forecasting the yearly mean demands and their dispersions are defined as innovations state space models [20]. They combine the seasonal, trend and error components in different ways (additively or multiplicatively). For each TS the optimal ETS model is selected using Akaike information criterion (AIC).

Ensembling is used for reduction the model variance related to the stochastic nature of SGD, and also related to data and parameter uncertainty. Ensembling

is seen as a much more powerful regularization technique than more popular alternatives, e.g. dropout or L2-norm penalty [21]. In our case, ensembling combines individual forecasts at three levels: stage of training level, data subset level and model level. At the stage of training level, the forecasts produced by L most recent training epochs are averaged. This can reduce the effect of stochastic searching, i.e. calming down the noisy SGD optimization process. At the data subset level, we use K models which learn on the subsets of the training set, $\Psi_1, \Psi_2, \dots, \Psi_K$. Each k -th model produces forecasts for TS included in its own training subset Ψ_k . Then the forecasts produced by the pool of K models are averaged individually for each TS. The third level of ensembling simply averages the forecasts for each TS generated in R independent runs of a pool of K models. In each run, the training subsets Ψ_k are created anew (see [1] for details).

2.2 Time Series Processing

A monthly electricity demand TS exhibits a trend, yearly seasonality and random component (see Fig. 2(a)). To simplify the forecasting problem, the TS is preprocessed as follows. Let $\{z_t\}_{t=1}^N$ be a monthly electricity demand TS starting from January and ending in December. This TS is divided into yearly subsequences $\{z_t^i\}_{t=12(i-1)+1}^{12(i-1)+12}, i = 1, \dots, N/12$. Each i -th subsequence is expressed by a vector $\mathbf{z}_i = [z_{i,1} z_{i,2} \dots z_{i,12}]^T$. The normalized version of $\mathbf{z}_i$, $\mathbf{y}_i = [y_{i,1} y_{i,2} \dots y_{i,12}]^T$, is determined as follows:

$$y_{i,j} = \frac{z_{i,j} - \bar{z}_i}{\sigma_i} + 1 \quad (2)$$

where $j = 1, \dots, 12$, $\bar{z}_i$ is a mean of subsequence $\{z_t^i\}$, and $\sigma_i = \sqrt{\sum_{j=1}^{12} (z_{i,j} - \bar{z}_i)^2}$ is a measure of its dispersion.

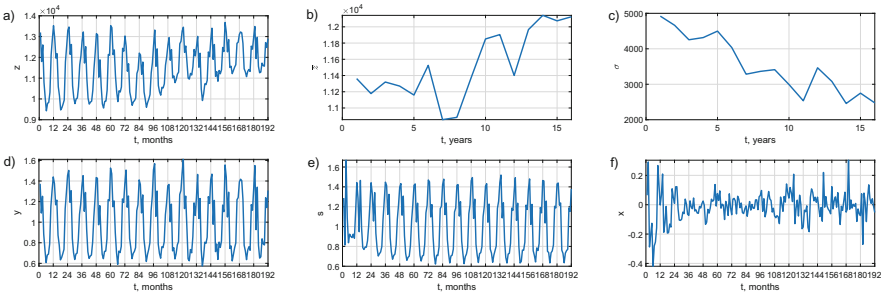


Fig. 2. TS preprocessing: (a) original TS $\{z_t\}$, (b) yearly mean demand TS $\{\bar{z}_t\}$, (c) yearly dispersion TS $\{\sigma_t\}$, (d) normalized TS $\{y_t\}$, (e) seasonal component TS $\{s_t\}$, and (f) normalized and deseasonalized TS $\{x_t\}$

Note that yearly subsequences $\{z_t^i\}$ have different means and dispersions (see Fig. 2(a)). After normalization they are unified, i.e. all yearly subsequences have

an average of one, the same variance and also a unity of length. They carry information about the shapes of the yearly sequences. Now we create a new TS composed of normalized subsequences representing successive yearly periods: $\{y_t\} = \{\mathbf{y}_i^T\}_{i=1}^{N/12} = \{y_{1,1}, y_{1,2}, \dots, y_{N/12,12}\}$. This TS is shown in Fig. 2(d). Note its regular character and stationarity.

The TS of the mean yearly demand, $\{\bar{z}_i\}_{i=1}^{N/12}$, and yearly dispersion, $\{\sigma_i\}_{i=1}^{N/12}$, are shown in Fig. 2(b) and (c), respectively. They are forecasted by ETS one step ahead (for the next year) and used for denormalization.

The normalized TS, $\{y_t\}$, is further deseasonalized. To do so, we use a simplified Holt-Winters multiplicative seasonal model with only one component:

$$s_{t+12} = \beta y_t + (1 - \beta)s_t \quad (3)$$

where s_t is the seasonal component at timepoint t and $\beta \in [0, 1]$ is a smoothing coefficient.

The seasonal component is shown in Fig. 2(e). It is used for deseasonalization during the on-the-fly preprocessing. The TS $\{y_t\}$ is deseasonalized in each training epoch using the updated values of seasonal components. These updated values are calculated from (3), where parameters, 12 initial seasonal components and β , are increasingly fine tuned for each TS in each epoch by SGD.

The TS is deseasonalized using rolling windows: input and output ones. The input window contains twelve consecutive elements of the TS which after deseasonalization will be the RD-LSTM inputs. The corresponding output window contains the next twelve consecutive elements, which after deseasonalization will be the RD-LSTM outputs. The TS fragments inside both windows are deseasonalized by dividing them by the relevant seasonal component. Then, to limit the destructive impact of outliers on the forecasts, a squashing function, $\log(\cdot)$, is applied. The resulting deseasonalization can be expressed as follows:

$$x_t = \log\left(\frac{y_t}{s_t}\right) \quad (4)$$

where x_t is the deseasonalized t -th element of the normalized TS, and s_t is the t -th seasonal component.

The preprocessed TS sequences contained in the successive input and output windows are represented by vectors: $\mathbf{x}_t^{in} = [x_t \dots x_{t+12}]$, $\mathbf{x}_t^{out} = [x_{t+13} \dots x_{t+24}]$, $t = 1, \dots, N - 24$. These vectors are included in the training subset for the i -th TS: $\Phi_i = \{(\mathbf{x}_t^{in}, \mathbf{x}_t^{out})\}_{t=1}^{N-24}$. The training subsets for all M TS are combined and form the training set $\Psi = \{\Phi_1, \dots, \Phi_M\}$ which is used for RD-LSTM cross-learning. Note the dynamic character of the training set. It is updated in each epoch because the seasonal components in (4) are updated by SGD.

The forecasts produced by RD-LSTM, $\hat{x}_t$, are reseasonalized as follows:

$$\hat{y}_t = s_t \exp(\hat{x}_t) \quad (5)$$

where s_t is determined from (3) on the basis of the TS history.

Finally, the TS is denormalized using transformed Eq. (2):

$$\hat{z}_{i,j} = (\hat{y}_{i,j} - 1)\hat{\sigma}_i + \hat{\hat{z}}_i \quad (6)$$

where i refers to the forecasted yearly period, $j = 1, \dots, 12$, $\hat{\hat{z}}_i$ and $\hat{\sigma}_i$ are the forecasted yearly mean and dispersion for period i .

2.3 Residual Delated LSTM

The RD-LSTM architecture used in this study is shown in Fig. 3(a) [1]. It is composed of four recurrent layers and a linear unit LU. The first layer consists of the standard LSTM block shown in Fig. 3(b). The subsequent three layers consist of RD-LSTM blocks, i.e. blocks equipped with dilated recurrent skip connections and a spatial shortcut path from lower layers (Fig. 3(c)).

A standard LSTM block consists of hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$. The cell state contains information learned from the previous time steps which 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, $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, and 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 input $\mathbf{x}_t$. The equations for a standard LSTM block are shown in Table 1.

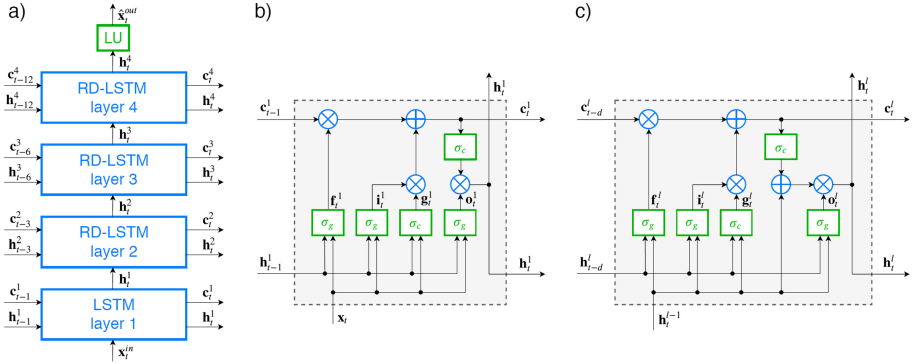


Fig. 3. RD-LSTM architecture (a), LSTM block (b), and RD-LSTM block (c)

The RD-LSTM blocks employ dilation mechanism proposed in [22]. It is to solve three main problems related to RNN learning on long sequences: complex dependencies, vanishing and exploding gradients, and efficient parallelization. It is characterized by multi-resolution dilated recurrent skip connections. To compute the current states of the LSTM block, the last $d - 1$ states are skipped, i.e. a dilated LSTM block receives as input states $\mathbf{c}_{t-d}$ and $\mathbf{h}_{t-d}$, where $d > 1$ is a dilation. Usually multiple dilated recurrent layers are stacked with hierarchical dilations to construct a system, which learns the temporal dependencies

Table 1. Equations for the forward pass of LSTM blocks

Standard LSTM block	RD-LSTM block
$\mathbf{f}_t^1 = \sigma_g(\mathbf{W}_f^1 \mathbf{x}_t + \mathbf{V}_f^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_f^1)$	$\mathbf{f}_t^l = \sigma_g(\mathbf{W}_f^l \mathbf{h}_t^{l-1} + \mathbf{V}_f^l \mathbf{h}_{t-d}^{l-1} + \mathbf{b}_f^l)$
$\mathbf{i}_t^1 = \sigma_g(\mathbf{W}_i^1 \mathbf{x}_t + \mathbf{V}_i^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_i^1)$	$\mathbf{i}_t^l = \sigma_g(\mathbf{W}_i^l \mathbf{h}_t^{l-1} + \mathbf{V}_i^l \mathbf{h}_{t-d}^{l-1} + \mathbf{b}_i^l)$
$\mathbf{g}_t^1 = \sigma_c(\mathbf{W}_g^1 \mathbf{x}_t + \mathbf{V}_g^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_g^1)$	$\mathbf{g}_t^l = \sigma_c(\mathbf{W}_g^l \mathbf{h}_t^{l-1} + \mathbf{V}_g^l \mathbf{h}_{t-d}^{l-1} + \mathbf{b}_g^l)$
$\mathbf{o}_t^1 = \sigma_g(\mathbf{W}_o^1 \mathbf{x}_t + \mathbf{V}_o^1 \mathbf{h}_{t-1}^1 + \mathbf{b}_o^1)$	$\mathbf{o}_t^l = \sigma_g(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{V}_o^l \mathbf{h}_{t-d}^{l-1} + \mathbf{b}_o^l)$
$\mathbf{c}_t^1 = \mathbf{f}_t^1 \otimes \mathbf{c}_{t-1}^1 + \mathbf{i}_t^1 \otimes \mathbf{g}_t^1$	$\mathbf{c}_t^l = \mathbf{f}_t^l \otimes \mathbf{c}_{t-d}^l + \mathbf{i}_t^l \otimes \mathbf{g}_t^l$
$\mathbf{h}_t^1 = \mathbf{o}_t^1 \otimes \sigma_c(\mathbf{c}_t^1)$	$\mathbf{h}_t^l = \mathbf{o}_t^l \otimes (\sigma_c(\mathbf{c}_t^l) + \mathbf{h}_{t-d}^{l-1})$

where $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{b}$ are input weights, recurrent weights and biases, respectively, σ_c is a hyperbolic tangent function, σ_g is a sigmoid activation function $(1 + e^{-x})^{-1}$, $\otimes$ denotes the Hadamard product, superscript 1 refers to the first layer of RD-LSTM network, where we use the standard LSTM block, superscript l indicates the layer number for RD-LSTM blocks (from 2 to 4 in our case), and d is a dilation (3, 6 or 12 in our case).

of different scales at different layers. In [22], it was shown that this solution can reliably improve the ability of recurrent models to learn long-term dependency. Dilated RNN can be particularly useful for seasonal TS. In this case dilations can be related to seasonality. In our case we use $d = 3, 6$ and 12 .

A residual LSTM was proposed in [23] to enable effective training of deep networks with multiple LSTM layers by avoiding vanishing or exploding gradients in the temporal domain. Residual LSTM provides a shortcut path between adjacent layer outputs. The shortcut paths are used to allow gradients to flow through a network directly, without passing through non-linear activation functions. In our implementation, we introduced shortcut paths extending equation for the hidden state (note additional component, $\mathbf{h}_t^{l-1}$, for a hidden state in the right column of Table 1, where the RD-LSTM computation process is shown).

A linear unit, LU, transforms the output of the last layer, $\mathbf{h}_t^4$, into the forecast of the output x-vector:

$$\hat{\mathbf{x}}_t^{out} = \mathbf{W}_x \mathbf{h}_t^4 + \mathbf{b}_x \quad (7)$$

Note that RD-LSTM works on 12-component x-vectors. It produces the forecasts for the whole yearly period receiving the previous yearly period as input. The parameters of RD-LSTM, i.e. input weights $\mathbf{W}$, recurrent weights $\mathbf{V}$, and biases $\mathbf{b}$, are learned using SGD in the cross-learning mode simultaneously with the ETSd parameters. The length of the cell and hidden states, m , the same for all layers, was selected on the training set to ensure the highest performance.

3 Results

The proposed forecasting model is applied for monthly electricity demand forecasting for 35 European countries. The real-world data are taken from the ENTSO-E repository (www.entsoe.eu). The TS lengths vary from 5 to 24 years.

The forecasting problem is to produce the forecasts for the twelve months of 2014 (last year of data) using data from the previous period for training. For hyperparameter selection the model learned on the TS fragments up to 2012, and then it was validated on 2013. The selected hyperparameters were used to build the model for 2014: number of epochs 10, learning rate 10^{-3} , length of the cell and hidden states $m = 40$, asymmetry parameter in pinball loss $\tau = 0.4$, ensembling parameters: $L = 5$, $K = 4$, $R = 3$. RD-LSTM was implemented in C++ relying on the DyNet library and run in parallel on an 8-core CPU. We employ R implementation of ETS (function `ets` from package `forecast`).

The proposed model was compared with its predecessor, ETS+RD-LSTM [1], and other state-of-the-art models based on ML as well as classical statistical models. They include: k -nearest neighbor weighted regression model, k -NNw, fuzzy neighborhood model, FNM, general regression NN model, GRNN, multilayer perceptron, MLP, adaptive neuro-fuzzy inference system, ANFIS, LSTM model, ARIMA model, and ETS model. All ML models were used also in +ETS versions, where the TS were initially normalized and the yearly mean and dispersion were forecasted using ETS (just like in this study). Details of the comparative models can be found in [18, 19, 24, 25].

Table 2 shows the forecast results averaged over 35 countries, i.e. median of absolute percentage error (APE), mean APE (MAPE), interquartile range of APE as a measure of the forecast dispersion, root mean square error (RMSE),

Table 2. Results comparison among proposed and comparative models

Model	Median <i>APE</i>	<i>MAPE</i>	<i>IQR</i>	<i>RMSE</i>	<i>MPE</i>
k-NNw	2.89	4.99	3.85	368.79	−1.87
FNM	2.88	4.88	4.26	354.33	−2.03
N-WE	2.84	5.00	3.97	352.01	−1.91
GRNN	2.87	5.01	4.02	350.61	−1.87
k-NNw+ETS	2.71	4.47	3.52	327.94	−1.25
FNM+ETS	2.64	4.40	3.46	321.98	−1.26
N-WE+ETS	2.68	4.37	3.36	320.51	−1.26
GRNN+ETS	2.64	4.38	3.51	324.91	−1.26
MLP	2.97	5.27	3.84	378.81	−1.37
MLP+ETS	3.11	4.80	4.12	358.07	−1.71
ANFIS	3.56	6.18	4.87	488.75	−2.51
ANFIS+ETS	3.54	6.32	4.26	464.29	−1.30
LSTM	3.73	6.11	4.50	431.83	−3.12
LSTM+ETS	3.08	5.19	4.54	366.45	−1.41
ARIMA	3.32	5.65	5.24	463.07	−2.35
ETS	3.50	5.05	4.80	374.52	−1.04
ETS+RD-LSTM	2.74	4.48	3.55	347.24	−1.11
3ETS+RD-LSTM	2.64	4.09	3.13	314.01	−0.32

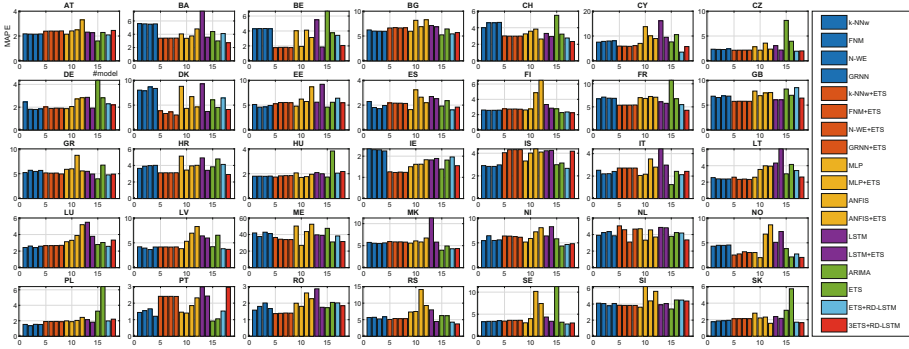


Fig. 4. MAPE for each country

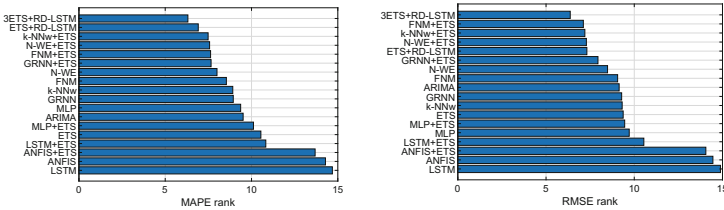


Fig. 5. Rankings of the models

and mean PE (MPE). The proposed model is denoted by 3ETS+RD-LSTM. As can be seen from this table, all error measures indicate that 3ETS+RD-LSTM is the most accurate model comparing with its competitors. It outperforms its predecessor, ETS+RD-LSTM, by 8.7% in MAPE and 9.5% in RMSE.

MPE provides information on potential forecast bias. All the models produced negatively biased forecasts, i.e. overpredicted. But for 3ETS+RD-LSTM, the t -test did not reject the null hypothesis that PE comes from a normal distribution with mean equal to zero (p -value = 0.44). All other models did not pass this test. So it can be concluded that 3ETS+RD-LSTM, as the only model, produced unbiased forecasts. Note that 3ETS+RD-LSTM has the mechanism to deal with bias. The loss function (1) asymmetry is controlled by parameter τ . It was selected as 0.4, which allowed the model to reduce the negative bias.

Figure 4 depicts more detailed results, MAPE for each country. As can be seen, in most cases 3ETS+RD-LSTM is one of the most accurate models. Figure 5 depicts the model rankings based on MAPE and RMSE. They show average ranks of the models in the rankings for individual countries. Note the first position of 3ETS+RD-LSTM in both rankings.

Examples of forecasts produced by the selected models are shown in Fig. 6. For PL and DE data, MAPE is on the low level around 2% while for GB data the forecasts are strongly underestimated, over 5%. This is because the demand

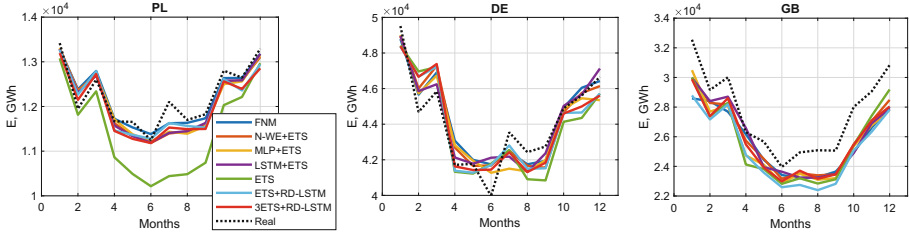


Fig. 6. Examples of forecasts produced by selected models

for GB went up unexpectedly in 2014 despite the downward trend observed in the previous period.

Summarizing experimental research, it should be noted that the forecasting model performance depends significantly on the appropriate TS preprocessing. Although LSTM deals with raw data, without preprocessing [19], introducing initial normalization and dynamic deseasonalization in 3ETS+RD-LSTM improved significantly LSTM performance.

It should be noted that LSTM based models are more complex than other comparative models. Due to the huge number of parameters and complicated learning procedure using backpropagation through time, the learning time of LSTM is much longer than for other comparative models.

4 Conclusion

In this work, we proposed an extended hybrid RD-LSTM and ETS model for MTLF. It combines initial TS decomposition into three components (normalized TS, trend, and dispersion), ETS modules for trend and dispersion forecasting, ETS for deseasonalization, advanced LSTM, and ensembling. The model has a hierarchical structure composed of a global part learned across many TS (LSTM) and a TS specific part (normalization and deseasonalization). Deseasonalization and LSTM are combined in a simultaneous learning process using SGD which leads to learning TS representations and mapping at the same time.

We used residual dilated LSTM, which can capture better long-term seasonal relationships and ensure more efficient training. This is because of dilated recurrent skip connections and a spatial shortcut path from lower layers. To deal with a forecast bias, an asymmetric pinball loss function was applied. Three-level ensembling provides regularization reducing the model variance, which has sources in the stochastic nature of SGD, and also in data and parameter uncertainty.

An experimental study, monthly electricity demand forecasting for 35 European countries, demonstrated the state-of-the-art performance of the proposed model. It generated more accurate forecasts than its predecessor (ETS+RD-LSTM), classical models such as ARIMA and ETS as well as state-of-the-art models based on ML.

References

1. Dudek, G., Pelka, P., Smyl, S.: A hybrid residual dilated LSTM end exponential smoothing model for mid-term electric load forecasting. arXiv preprint [arXiv:2004.00508](https://arxiv.org/abs/2004.00508) (2020)
2. Suganthi, L., Samuel, A.-A.: Energy models for demand forecasting - a review. *Renew. Sust. Energ. Rev.* **16**(2), 1223–1240 (2002)
3. Barakat, E.H.: Modeling of nonstationary time-series data. Part II. Dynamic periodic trends. *Int. J. Elec. Power* **23**, 63–68 (2001)
4. González-Romera, E., Jaramillo-Morán, M.-A., Carmona-Fernández, D.: Monthly electric energy demand forecasting with neural networks and Fourier series. *Energ. Convers. Manage.* **49**, 3135–3142 (2008)
5. Chen, J.F., Lo, S.K., Do, Q.H.: Forecasting monthly electricity demands: an application of neural networks trained by heuristic algorithms. *Information* **8**(1), 31 (2017)
6. Gavrilas, M., Ciutea, I., Tanasa, C.: Medium-term load forecasting with artificial neural network models. In: *IEEE Conference on Electricity Distribution Publication*, vol. 6 (2001)
7. Doveh, E., Feigin, P., Hyams, L.: Experience with FNN models for medium term power demand predictions. *IEEE Trans. Power Syst.* **14**(2), 538–546 (1999)
8. Pelka, P., Dudek, G.: Medium-term electric energy demand forecasting using generalized regression neural network. In: Świątek, J., Borzemski, L., Wilimowska, Z. (eds.) *ISAT 2018. AISC*, vol. 853, pp. 218–227. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-99996-8_20
9. Pei-Chann, C., Chin-Yuan, F., Jyun-Jie, L.: Monthly electricity demand forecasting based on a weighted evolving fuzzy neural network approach. *Int. J. Elec. Power* **33**, 17–27 (2011)
10. Ahmad, T., Chen, H.: Potential of three variant machine-learning models for forecasting district level medium-term and long-term energy demand in smart grid environment. *Energy* **160**, 1008–1020 (2018)
11. Hewamalage, H., Bergmeir, C., Bandara, K.: Recurrent neural networks for time series forecasting: current status and future directions. arXiv preprint [arXiv:1909.00590v3](https://arxiv.org/abs/1909.00590v3) (2019)
12. Yan, K., Wang, X., Du, Y., Jin, N., Huang, H., Zhou, H.: Multi-step short-term power consumption forecasting with a hybrid deep learning strategy. *Energies* **11**(11), 3089 (2018)
13. Bedi, J., Toshniwal, D.: Empirical mode decomposition based deep learning for electricity demand forecasting. *IEEE Access* **6**, 49144–49156 (2018)
14. Zheng, H., Yuan, J., Chen, L.: Short-term load forecasting using EMD-LSTM neural networks with a XGBoost algorithm for feature importance evaluation. *Energies* **10**(8), 1168 (2017)
15. Narayan, A., Hipel, K.-W.: Long short term memory networks for short-term electric load forecasting. In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2573–2578 (2017)
16. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The M4 competition: results, findings, conclusion and way forward. *Int. J. Forecast.* **34**(4), 802–808 (2018)
17. Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *Int. J. Forecast.* **36**(1), 75–85 (2020)
18. Dudek, G., Pelka, P.: Pattern similarity-based machine learning methods for mid-term load forecasting: a comparative study. arXiv preprint [arXiv:2003.01475](https://arxiv.org/abs/2003.01475) (2020)

19. Pelka, P., Dudek, G.: Pattern-based long short-term memory for mid-term electrical load forecasting. In: 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, pp. 1–8 (2020). <https://doi.org/10.1109/IJCNN48605.2020.9206895>
20. Hyndman, R.-J., Koehler, A.-B., Ord, J.-K., Snyder, R.-D.: Forecasting with Exponential Smoothing: The State Space Approach. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-71918-2>
21. Oreshkin, B.-N., Carпов, D., Chapados, N., Bengio, Y.: N-BEATS: neural basis expansion analysis for interpretable time series forecasting. arXiv preprint [arXiv:1905.10437v4](https://arxiv.org/abs/1905.10437v4) (2020)
22. Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., et al.: Dilated recurrent neural networks. arXiv preprint [arXiv:1710.02224](https://arxiv.org/abs/1710.02224) (2017)
23. Kim, J., El-Khamy, M., Lee, J.: Residual LSTM: design of a deep recurrent architecture for distant speech recognition. arXiv preprint [arXiv:1701.03360](https://arxiv.org/abs/1701.03360) (2017)
24. Pelka, P., Dudek, G.: Pattern-based forecasting monthly electricity demand using multilayer perceptron. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2019. LNCS (LNAI), vol. 11508, pp. 663–672. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20912-4_60
25. Pelka, P., Dudek, G.: Neuro-fuzzy system for medium-term electric energy demand forecasting. In: Borzowski, L., Świątek, J., Wilimowska, Z. (eds.) ISAT 2017. AISC, vol. 655, pp. 38–47. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67220-5_4

ES-dRNN: A Hybrid Exponential Smoothing and Dilated Recurrent Neural Network Model for Short-Term Load Forecasting

Slawek Smyl¹, Grzegorz Dudek², and Paweł Pełka²

Abstract—Short-term load forecasting (STLF) is challenging due to complex time series (TS) which express three seasonal patterns and a nonlinear trend. This article proposes a novel hybrid hierarchical deep-learning (DL) model that deals with multiple seasonality and produces both point forecasts and predictive intervals (PIs). It combines exponential smoothing (ES) and a recurrent neural network (RNN). ES extracts dynamically the main components of each individual TS and enables on-the-fly deseasonalization, which is particularly useful when operating on a relatively small dataset. A multilayer RNN is equipped with a new type of dilated recurrent cell designed to efficiently model both short and long-term dependencies in TS. To improve the internal TS representation and thus the model's performance, RNN learns simultaneously both the ES parameters and the main mapping function transforming inputs into forecasts. We compare our approach against several baseline methods, including classical statistical methods and machine learning (ML) approaches, on STLF problems for 35 European countries. The empirical study clearly shows that the proposed model has high expressive power to solve nonlinear stochastic forecasting problems with TS including multiple seasonality and significant random fluctuations. In fact, it outperforms both statistical and state-of-the-art ML models in terms of accuracy.

Index Terms—Deep learning (DL), exponential smoothing (ES), hybrid forecasting models, recurrent neural networks (RNNs), short-term load forecasting (STLF), time series (TS) forecasting.

I. INTRODUCTION

ELECTRICITY demand forecasting for different horizons and granularity is an integral part of power system control, scheduling, and planning. Thus, it is extremely important for energy suppliers, system operators, financial institutions, and other participants in electric energy generation, transmission, distribution, and markets. At a short-term level, i.e., with a horizon from 1 h to seven days ahead and hourly granularity or less, electricity demand forecasting is the basis of power system operation including unit commitment, generation dispatch, hydro scheduling, hydrothermal coordination, spinning reserve allocation, interchange and low flow evaluation,

security assessment, and network diagnosis [1]. Modern power systems pose new challenges for the forecasting models due to issues connected with volatile distributed energy resources, integration of intermittent renewable energy resources, and deployment of demand-side management. As electricity demand is the primary driver of electricity prices, short-term load forecasting (STLF) plays a key role in competitive energy markets. The accuracy of forecasts translates directly into the financial performance of energy market participants. A related study revealed that a 1% reduction in forecasting error for a 10 GW utility can save up to \$1.6 million annually [2].

A. Related Work

The importance of STLF for the safe, reliable, and efficient operation of power systems as well as the complexity of the problem translates into great interest from researchers in this field. Nonlinear trends, multiple seasonality, variable variance and daily profile, and random fluctuations, make STLF challenging and place high demands on forecasting models. STLF approaches can be divided into three categories: statistical or econometric models, ML models, and hybrid ones. The first category includes auto-regressive integrated moving average (ARIMA) [3], exponential smoothing (ES) [4], linear regression [5], and Kalman filtering [6].

The main problem with many statistical STLF methods is their linear nature, which limits the implementation of nonlinear system dynamics. To extend the model's capabilities to approximate nonlinear relationships, local modeling is used. For example, in [7], the linear state-space model is learned progressively from the data using a Kalman filter. The method works by assuming temporally local linearity, which can be seen as an approximation of an underlying nonlinearity, generalizing the standard linear-Gaussian model with static parameters. In [8], the target nonlinear function was modeled locally in the neighborhood around the query pattern using linear regression. Due to initial data normalization, which simplified relationships between input and output data, linear models, such as partial least-squares regression, were able to compete with more sophisticated ML models.

Another problem with statistical methods is their limited ability to model complicated seasonal patterns. Standard ARIMA and Holt-Winters models can be extended to multiple seasonality [9] but they assume that the cycle shapes are all the same. In practice, the seasonal patterns

Manuscript received 29 November 2021; revised 24 November 2022; accepted 13 March 2023. Date of publication 31 August 2023; date of current version 6 August 2024. (Corresponding author: Grzegorz Dudek.)

Slawek Smyl is with Facebook, Menlo Park, CA 94025 USA (e-mail: slawek.smyl@gmail.com).

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

Digital Object Identifier 10.1109/TNNLS.2023.3259149

can greatly differ from each other (in STLF, daily cycles for workdays are usually significantly different from those for weekends). Considering the changing seasonal pattern requires a significant extension of the model. For example, in [10], an ES state space model was combined with Fourier terms, a Box-Cox transformation, and ARMA error correction. Extending the regression model with Fourier terms (harmonic regression) is a popular method of introducing seasonal components into statistical models [11]. Another approach to deal with seasonality is time series (TS) decomposition. Products of decomposition are less complex than the original TS and can be modeled using simpler models [8], [12], [13].

Other drawbacks statistical methods suffer from are limited adaptability, a shortage of expressive power, problems with capturing long-term dependencies, and introducing exogenous variables into the model. ML methods offer many more possibilities than statistical ones. They provide forecasting models with the ability to learn historical patterns and anomalies and successively improve prediction accuracy. The most researched ML models in the field of forecasting are neural networks (NNs) [14]. They can flexibly model complex nonlinear relationships between variables and reflect process variability in uncertain dynamic environments due to their universal approximation property. At the same time, NNs have their limitations such as disruptive and unstable training, the need for careful feature engineering, local optimality, weak interpretability, difficulty in matching the network architecture to the problem solved, tendency to overfitting, weak extrapolation ability and many parameters to estimate. These issues as well as problems with modeling complex seasonal patterns are addressed in STLF literature in various ways. For example, in [15], TS with multiple seasonality were represented by patterns of the daily profiles, which simplified greatly the forecasting problem. As a result, it was possible to use simpler, resistant-to-overfitting neural models with a small number of parameters. Among the NN architectures compared in [15] are multilayer perceptron (MLP), radial basis function (RBF) NN, generalized regression NN (GRNN), fuzzy counterpropagation NN, and self-organizing maps. Within the group, GRNN and MLP turned out to be the most accurate. A Bayesian approach was used in [16] to control MLP complexity and to select input variables. The Bayesian framework offered ways to avoid overfitting by regularization, to decide on the number of neurons by comparing the model evidence, and to deal with the inputs by soft-pruning. Many NN solutions for STLF combine the neural model, optimization method for hyperparameter selection and learning, and TS decomposition or a feature engineering method. An example can be found in [17] where TS is decomposed using wavelet transform to extract relevant information from the load curve, and MLP weights are adjusted using a particle swarm optimization algorithm. The most popular and universal NN, MLP, was recently replaced by randomized NN in STLF [18]. When a pattern-based representation is used, randomized NN can produce more accurate forecasts than MLP while having many advantages over MLP. These include extremely fast and easy training, simple architecture, a small number of hyperparameters and parameters to estimate, and ease of implementation.

NN architectures proposed for STLF in recent years are dominated by deep learning (DL) and recurrent NNs (RNN). The success of DL can be largely attributed to increased model complexity and the ability to cross-learn on massive datasets. This strengthens expressive power and the ability to extract patterns across multiple examples. DL architectures are composed of combinations of basic structures, such as MLPs, convolutional NNs (CNNs), and RNNs. New ideas in the field of DL have been successfully applied to STLF. Some examples are: [19], where deep residual NNs were proposed and applied to probabilistic load forecasting using Monte Carlo dropout; [20], where a multivariate fuzzy TS was converted into multichannel images and processed by CNN to produce load forecasts; [21], where an improved deep belief network for STLF considering demand-side management was proposed; and [22], where an STLF problem for individual residential households was addressed using long-short term memory (LSTM) RNN. Among NN forecasting models, modern RNNs such as LSTM and gated recurrent unit (GRU) are distinguished by their ability to model both short and long-term dependencies in TS. Therefore, they are readily used for STLF [23].

To improve further forecasting model performance, ensemble, and hybrid solutions have been developed. Ensembling is a reliable approach to increasing the forecast accuracy and robustness of both statistical and ML models. It combines, in some way, multiple models to produce a common response, controlling a bias-variance-covariance tradeoff [24]. Ensemble strategies for STLF take many forms. For example, in [25], an ensemble of NNs is proposed, which are trained on the products of wavelet decomposition; in [26], an empirical mode decomposition is applied to decompose the original interval-valued STLF data, and an ensemble of LSTMs is utilized to synchronously forecast and combine the components; and in [27], a data-driven multiobjective evolutionary ensemble learning is proposed with random vector functional link NNs as base learners. Hybrid approaches combine two or more methods in a common model, taking advantage of their strengths and avoiding their drawbacks. For example, statistical methods can help in data preprocessing and reduce the overfitting of ML models. Examples of model hybridization for STLF can be found in [28] where a temporal CNN is utilized to extract hidden information and long-term temporal relationships in the input data and a boosted tree model (LightGBM) is used to predict future loads based on the extracted features, and in [29], where both LSTM and wavelet decomposition extract TS features, on which an ensemble of RBF NNs is trained. The produced forecasts are aggregated by a localized generalization error model, which optimizes the ensemble member weights.

B. Motivation and Contribution

The motivation behind this work is threefold. First, STLF is extremely important for power system operation and energy market functioning. Forecast accuracy directly translates into the safe, reliable, and efficient operation of power systems as well as improved financial performance of energy

market participants. Second, STLF is a challenging problem due to complex three-component seasonality, nonlinear trend, significant stochastic component, and changing seasonal patterns. It requires a flexible forecasting model capable of capturing long-term and short-term dependencies in TS. Third, new advances in ML and DL, especially in sequential data processing, encourage their application in complex forecasting problems such as STLF. Modern RNNs can deal with multiple seasonality and long-term dependencies in TS. Hybrid solutions utilizing both statistical and DL methods improve representation learning and the exploration of hidden patterns. In this study, we extend our recent works [30] and [31], where we used a combined ES and LSTM model for forecasting TS with single and double seasonality. It is worth noting that the hybrid model proposed in [30] won the renowned M4 forecasting competition in 2018, outperforming a wide variety of state-of-the-art models. The winning model produced both the most accurate forecasts and the most precise PIs for 100 000 real-life TS. It was close to 10% more accurate than the benchmark ensemble model, which is a huge improvement [32]. This means the model has been reliably and rigorously verified on a wide range of forecasting problems. In this study, based on the main concept of the winning solution, we develop a new model specifically for an STLF problem with three seasonal patterns.

Our research contributions can be summarized as follows.

- 1) We propose a new dilated recurrent cell, dRNNCell, as a building block of dilated RNNs designed especially for STLF to deal with both short and long-term dependencies in TS.
- 2) We develop a new hybrid forecasting model for STLF combining ES and RNN. The model produces point and probabilistic forecasts in the form of PIs. It does not require initial TS decomposition and, due to its internal mechanisms such as adaptive TS preprocessing, cross-learning, and multiple dilations, can deal with complex TS expressing nonlinear trends, varying variance, and multiple seasonality. Our model is available as open-source code in the GitHub repository [33].
- 3) We propose a new mechanism for dynamically adjusting the smoothing coefficients used by ES. These coefficients are learned by RNN simultaneously with the main mapping function transforming inputs into forecasts to ensure optimal internal representation of TS and finally maximize the accuracy of the model.
- 4) We introduce a new three-component loss function based on pinball loss to optimize both the point forecasts and PIs.
- 5) We empirically demonstrate on real-world data for 35 European countries that the proposed hybrid model outperforms in STLF well-established statistical and state-of-the-art ML approaches.

The rest of the work is organized as follows. Section II describes the STLF data and defines the STLF problem. Section III presents the proposed forecasting model: its architecture, components, and features. The experimental framework used to evaluate the proposed model is described in Section IV. Finally, Section V concludes the work.

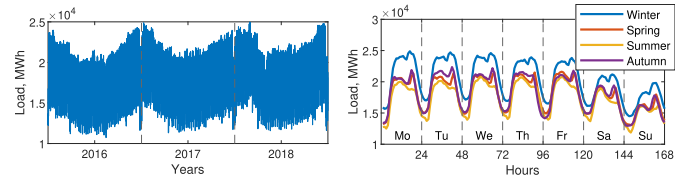


Fig. 1. Hourly electricity demand TS for Poland.

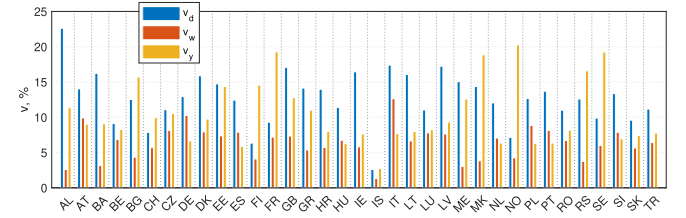


Fig. 2. Coefficients of daily, weekly, and yearly variations of electricity demand.

II. STLF PROBLEM AND DATA

In this study, we consider a univariate STLF problem where the task is to forecast future values of the hourly electricity demand TS for the next day, $\{z_\tau\}_{\tau=M+1}^{M+24}$, given a sequence of past observations, $\{z_\tau\}_{\tau=1}^M$. The problem is challenging because the electricity demand TS exhibits a trend, three types of variability: the daily, weekly, and yearly ones, and random fluctuations—see Fig. 1, where hourly electricity demand for the Polish power system is shown. The level of electricity demand and its long-term trend depend on a country's economic development and growth rate. One of the most important factors that can upset electricity demand in the short-term perspective are extreme weather conditions.

A very important issue from the point of view of power system control and planning is electricity demand variability observed in daily, weekly, and yearly periods. Fig. 2 shows variation coefficients for 35 European countries defined as $v = 100s/\bar{z}$. For daily electricity demand variations, v_d , $\bar{z}$, and s express the daily mean and standard deviation, for weekly variations, v_w , express the weekly mean and standard deviation of daily means, and for yearly variations, v_y , express the yearly mean and standard deviation of weekly means. Greater demand variability requires greater flexibility in generating units and the entire power system. As can be seen from Fig. 2, the lowest demand variations are for Iceland. The strongest daily variations ($v_d > 17\%$) are for Albania, Italy, Latvia, and Great Britain, while the strongest yearly variations ($v_y > 18\%$) are for Norway, France, Sweden, and Macedonia. The weekly variations are usually weaker than the daily and yearly ones, $v_w < 13\%$. Countries with the strongest weekly variations are Italy, Germany, Austria, and Poland. It is worth noting that the electricity demand variation changes over time, which is an additional challenge for forecasting models.

The electricity demand seasonalities are related to local climate, weather variability, and the types of consumers. Intensities of the seasonal fluctuations can be identified using harmonic analysis. Based on Parseval's theorem, the variance of a TS can be expressed by the sum of squares of its harmonic amplitudes. The contribution of the i th harmonic to the vari-

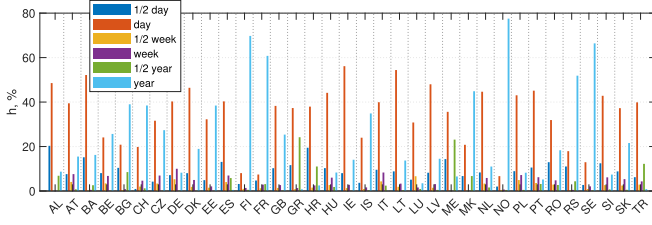


Fig. 3. Contribution of the most important harmonics in the TS variance.

ance can be expressed by the ratio $h_i = 100A_i^2/(2\text{Var}(z_t))$, where A_i is the i th harmonic amplitude and $\text{Var}(z_t)$ is the TS variance. Fig. 3 shows ratio h_i for the most important harmonics for 35 European countries. Note that for some countries the yearly seasonality strongly dominates compared to others ($h > 60\%$). These countries include Finland, France, Norway, and Sweden. Another extremely important seasonality is the daily one. The countries with the strongest daily seasonality ($h > 50$) are Bosnia and Herzegovina, Ireland, and Lithuania. In contrast, the weekly seasonality is less distinct with h below 10% for all countries. The highest weekly fluctuations ($h > 7\%$) are shown by Germany, Italy, Austria, and Poland. Some countries demonstrate stronger half-yearly seasonality than yearly ones. They include southern European countries such as Spain, Greece, Croatia, Italy, Montenegro, and Turkey. The half-yearly seasonality is related to the tourism industry, which increases energy demand in the summer season.

As can be seen from Fig. 1, the daily patterns for Tuesday through Friday from the same period of the year are similar, while those for Monday, Saturday, and Sunday are distinct. The daily shapes are dependent on the period of the year and can vary over the years. High similarity in daily shapes makes forecasting easier. Fig. 4 shows boxplots for distances between daily patterns representing the same days of the neighboring weeks, $d_t = \|\hat{\mathbf{z}}_t - \hat{\mathbf{z}}_{t+7}\|_2$. The daily pattern is defined as the centered and normalized daily vector: $\hat{\mathbf{z}}_t = (\mathbf{z}_t - \bar{\mathbf{z}}_t)/\|\mathbf{z}_t - \bar{\mathbf{z}}_t\|_2$, where $\mathbf{z}_t = [z_{t,1}, \dots, z_{t,24}]$ is the vector of hourly demands for the t th day and $\bar{\mathbf{z}}_t$ is the mean demand for that day. Vectors $\hat{\mathbf{z}}_t$ have mean zero, the same variance and unity length. The daily profiles expressed by $\hat{\mathbf{z}}_t$ differ only in shape. From Fig. 4, we can observe that the most similar profiles are for Lithuania, Poland, Germany, and Ireland ($0.0030 < d < 0.0033$), while the most dissimilar ones are for Iceland, Luxembourg, and Switzerland ($0.015 < d < 0.018$).

III. FORECASTING MODEL

A block diagram of the proposed forecasting model is shown in Fig. 5. The model is trained in a cross-learning mode [30], i.e., simultaneously on L hourly electricity load TS. Input Z represents a set of L TS: $\{\{z_\tau^l\}_{\tau=1}^{M_l}\}_{l=1}^L$, where M_l is an l th TS length. Output $\hat{Z}$ is a set of L forecasts of the daily sequences $\{\{\hat{z}_\tau^l\}_{\tau=M_l+1}^{M_l+24}\}_{l=1}^L$. An ES component expresses each TS from Z by two smoothing equations, i.e., for level and seasonality. The seasonal components, S , are used by the preprocessing component to deseasonalize the TS. This component also normalizes and squashes the TS and prepares

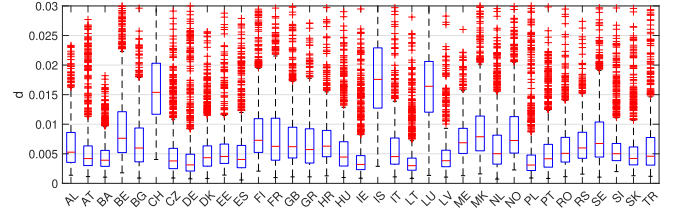


Fig. 4. Boxplots of distances between daily patterns of electricity demand.

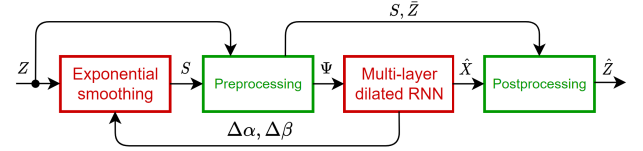


Fig. 5. Block diagram of the proposed forecasting system.

training sets Ψ for RNN learning. It feeds the processing parameters, i.e., seasonal components S and average values of TS sequences $\bar{Z}$, to the postprocessing component. RNN produces forecasts for the deseasonalized, normalized, and squashed daily sequences and their PIs for each TS ($\hat{X}$). These forecasts are postprocessed to obtain forecasts in real values, $\hat{Z}$. RNN also produces corrections of the ES smoothing parameters, $\Delta\alpha$ and $\Delta\beta$, to tune them properly. In the ensemble version, the model is trained E times and the forecasts are averaged. The diversity of ensemble learners, which decides about ensemble learning success [24], is achieved by random initial parameters.

Details of the model are described below.

A. ES Component

As shown in Section II, the hourly electricity load TS, $\{z_\tau\}_{\tau=1}^M$, exhibits complex phenomena with three seasonalities. To deal with the challenging STLF problem, the TS is deseasonalized, normalized, and squashed. Then, it is predicted by RNN. Deseasonalization is performed using a seasonal component produced by a simplified Holt-Winters multiplicative seasonal model in the form

$$\begin{aligned} l_\tau &= \alpha \frac{z_\tau}{s_\tau} + (1 - \alpha)l_{\tau-1} \\ s_{\tau+168} &= \beta \frac{z_\tau}{l_\tau} + (1 - \beta)s_\tau \end{aligned} \quad (1)$$

where l_τ is a level component, s_τ is a weekly seasonal component, and $\alpha, \beta \in [0, 1]$ are smoothing coefficients.

The series are hourly and exhibit daily, weekly, and yearly seasonalities. The (1) includes only weekly seasonality. However, the daily seasonality (24 h values) is part of the weekly seasonality (168 h values). In addition, the series are processed in 24-h steps, so the daily seasonality is to some extent “escaped”—the RNN always learns to forecast a whole day starting from midnight. The yearly seasonality impact is dealt with partly by normalization and partly by using date-related regressors, month, and week of the year.

A unique feature of the model is that the smoothing coefficients are learned by RNN. In addition to predicting

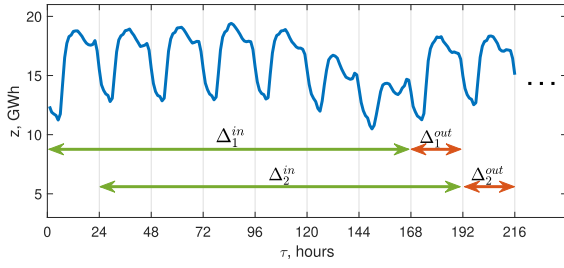


Fig. 6. Moving windows used for preprocessing TS z_τ .

the TS sequence and its PI, RNN also predicts corrections for smoothing coefficients, $\Delta\alpha_t$ and $\Delta\beta_t$. The smoothing coefficients are adapted in each recursive step t using the following corrections:

$$\begin{aligned}\alpha_{t+1} &= \sigma(I\alpha + \Delta\alpha_t) \\ \beta_{t+1} &= \sigma(I\beta + \Delta\beta_t)\end{aligned}\quad (2)$$

where $I\alpha$ and $I\beta$ are initial values of the smoothing coefficients (hyperparameters), and σ is a sigmoid function, which maintains the coefficients within a range from 0 to 1.

The smoothing coefficients have a dynamic character. This is because the corrections produced by RNN in each recursive step depend on current and past TS characteristics (shape, level, and seasonal pattern) and time variables that indicate in what phase of the weekly, monthly, and yearly cycles the predicted daily sequence is (see RNN input pattern (6)). The dynamic Holt-Winters equations take the form

$$\begin{aligned}l_{t,\tau} &= \alpha_t \frac{z_\tau}{s_{t,\tau}} + (1 - \alpha_t)l_{t,\tau-1} \\ s_{t,\tau+168} &= \beta_t \frac{z_\tau}{l_{t,\tau}} + (1 - \beta_t)s_{t,\tau}.\end{aligned}\quad (3)$$

B. Preprocessing and Postprocessing Components

To prepare input and output data for RNN we use two adjacent moving windows: input window Δ^{in} of size 168 h and output window Δ^{out} of size 24 h. The input window covers a weekly period to expose the RNN to the specific features of the series in this period directly. The output window covers the forecast daily sequence.

The windows are shifted by 24 h to obtain subsequent input and output patterns (see Fig. 6), which are defined as

$$\begin{aligned}\mathbf{x}_1^{\text{in}} &= [x_1, \dots, x_{168}], & \mathbf{x}_1^{\text{out}} &= [x_{169}, \dots, x_{192}] \\ \mathbf{x}_2^{\text{in}} &= [x_{25}, \dots, x_{192}], & \mathbf{x}_2^{\text{out}} &= [x_{193}, \dots, x_{216}] \\ & \dots & & \dots\end{aligned}\quad (4)$$

where the t th pair of patterns represent deseasonalized, normalized, and squashed TS sequences covered by the t th pair of windows

$$x_\tau = \log \frac{z_\tau}{\bar{z}_t \hat{s}_{t,\tau}} \quad (5)$$

where $\tau \in \Delta_t^{\text{in}} \cup \Delta_t^{\text{out}}$, i.e., $\tau \in [24(t-1)+1, 24(t-1)+192]$, $\bar{z}_t$ is the average TS value in the t th input window, i.e., $\bar{z}_t = 1/168 \sum_{\tau=24(t-1)+1}^{24(t-1)+168} z_\tau$, and $\hat{s}_{t,\tau}$ is the seasonal component determined using (3) for recursive step t (in this step t th pair of patterns (4) are used for RNN training).

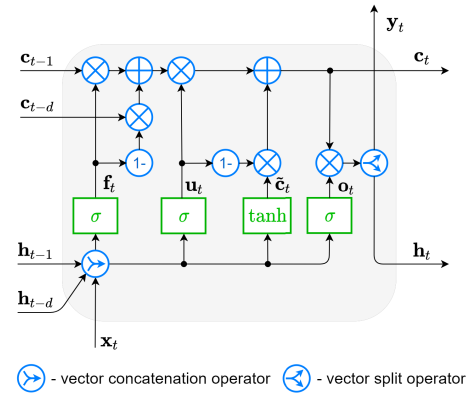


Fig. 7. Proposed dRNNCell.

TS sequences are squashed using a log function to prevent outliers from upsetting the learning process. Note that in (5), seasonal component $s_{t,\tau}$ is adapted for each t th pattern in each training epoch. Thus, the training set has a dynamic character. It is updated on-the-fly during learning. This process can be seen as the search for the optimal representation for RNN.

To introduce more input information related to the forecast sequence, the input patterns are extended as follows:

$$\mathbf{x}_t^{\text{in}'} = [\mathbf{x}_t^{\text{in}}, \hat{\mathbf{s}}_t, \log_{10}(\bar{z}_t), \mathbf{d}_t^w, \mathbf{d}_t^m, \mathbf{d}_t^y] \quad (6)$$

where $\hat{\mathbf{s}}_t$ is a vector of 24 seasonal components predicted by ES for the output period t reduced by 1, i.e., $\hat{\mathbf{s}}_t = [\hat{s}_{t,\tau-1}]_{\tau=24(t-1)+169}^{24(t-1)+192}$, $\mathbf{d}_t^w \in \{0, 1\}^7$, $\mathbf{d}_t^m \in \{0, 1\}^{31}$, and $\mathbf{d}_t^y \in \{0, 1\}^{52}$ are binary one-hot vectors encoding day of the week, day of the month, and week of the year for the forecast day, respectively.

Vectors $\mathbf{d}_t^w$ and $\mathbf{d}_t^y$ inform about the location of the forecast sequence in the weekly and yearly cycles, $\mathbf{d}_t^m$ helps to deal with fixed-date public holidays, $\log_{10}(\bar{z}_t)$ informs about the level of the TS (squashing function matches the level range with the range of the other components of $\mathbf{x}_t^{\text{in}'}$), and $\hat{\mathbf{s}}_t$ introduces additional information about the daily variability.

RNN is trained on training samples $(\mathbf{x}_t^{\text{in}'}, \mathbf{x}_t^{\text{out}'})$ (updated in each recursive step t), and produces forecasts of the output patterns $\hat{\mathbf{x}}_t^{\text{out}'} = [\hat{x}_\tau]_{\tau \in \Delta_t^{\text{out}'}}$ and their quantiles defining PIs. The postprocessing component converts these forecasts to real value forecasts using transformed (5)

$$\hat{z}_\tau = \exp(\hat{x}_\tau) \bar{z}_t \hat{s}_{t,\tau}. \quad (7)$$

C. RNN Component

RNN employs a new type of gated recurrent cell, dilated RNN cell (dRNNCell), which is shown in Fig. 7. It is derived from the LSTM [34] and GRU [35] cells. It is designed to operate as part of a multilayer dilated RNN [36] and as in [37] its output 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.

The cell uses two states, c -state (also called a cell state), which is close to the standard LSTM or GRU state, and h -state, which is the controlling state (also called a hidden state). At each time step t , the whole dRNNCell input is a concatenation of $\mathbf{x}_t$, $\mathbf{h}_{t-1}$, and $\mathbf{h}_{t-d}$, where $\mathbf{x}_t$ is a standard

input at time t (either from a previous layer or an input to the RNN), $\mathbf{h}_{t-1}$ is the most recent h -state, and $\mathbf{h}_{t-d}$ is the delayed state ($d > 1$). Both c - and h -states are saved in a list, to be used as delayed states. 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 dRNNCell uses the following gates: fusion (f), update (u), and output (o) gates. All the gates transform nonlinearly input vectors $\mathbf{x}_t$, $\mathbf{h}_{t-1}$, and $\mathbf{h}_{t-d}$ using sigmoid function (σ). A candidate c -state, $\tilde{\mathbf{c}}_t$, is produced by transforming input vectors using tanh nonlinearity. All nonlinear transformations of the input vectors are as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{V}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{h}_{t-d} + \mathbf{b}_f) \quad (8)$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_u \mathbf{x}_t + \mathbf{V}_u \mathbf{h}_{t-1} + \mathbf{U}_u \mathbf{h}_{t-d} + \mathbf{b}_u) \quad (9)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{V}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{h}_{t-d} + \mathbf{b}_o) \quad (10)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{V}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{h}_{t-d} + \mathbf{b}_c) \quad (11)$$

where $\mathbf{W}$, $\mathbf{V}$, and $\mathbf{U}$ are weight matrices, and $\mathbf{b}$ are bias vectors.

The c -state is a weighted combination of past c -states and new candidate state $\tilde{\mathbf{c}}_t$ computed in the current step

$$\mathbf{c}_t = \mathbf{u}_t \otimes (\mathbf{f}_t \otimes \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \otimes \mathbf{c}_{t-d}) + (1 - \mathbf{u}_t) \otimes \tilde{\mathbf{c}}_t \quad (12)$$

where $\otimes$ denotes the Hadamard product (element-wise product).

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.

The controlling state and the output of the cell are calculated based on the new c -state and output gate as follows:

$$\mathbf{h}'_t = \mathbf{o}_t \otimes \mathbf{c}_t \quad (13)$$

$$\mathbf{y}_t = [h'_{t,1}, \dots, h'_{t,s_y}] \quad (14)$$

$$\mathbf{h}_t = [h'_{t,s_y+1}, \dots, h'_{t,s_y+s_h}]. \quad (15)$$

The new features of dRNNCell can be summarized as follows.

- 1) dRNNCells are fed by both recent states, $\mathbf{c}_{t-1}$, $\mathbf{h}_{t-1}$, and delayed states, $\mathbf{c}_{t-d}$, $\mathbf{h}_{t-d}$, $d > 1$. Delayed states introduce the lagged information from d steps back to facilitate seasonal and long-term modeling.
- 2) The cell includes two internal weighting mechanisms for c -states, which are controlled by f - and u -gates, respectively. The first weights recent and delayed states, while the second weights the past and candidate states. The purpose of this mechanism is to improve the introduction of new information into the c -state and eliminate old information from the c -state.
- 3) Instead of the typical one output, the cell has two outputs: $\mathbf{y}_t$, which passes the forecast information to the next layer, and $\mathbf{h}_t$, which contains information for controlling the gates in the next step. This split allows these outputs to be better adapted to their distinct functions.

The dRNNCell is part of a multilayer dilated RNN, which is composed of a number of blocks, each composed of one or more cells. In Fig. 8, there are two blocks, the first with two layers dilated 2 and 7, respectively, and the second, just with a single layer dilated 4. Dilated RNN architecture

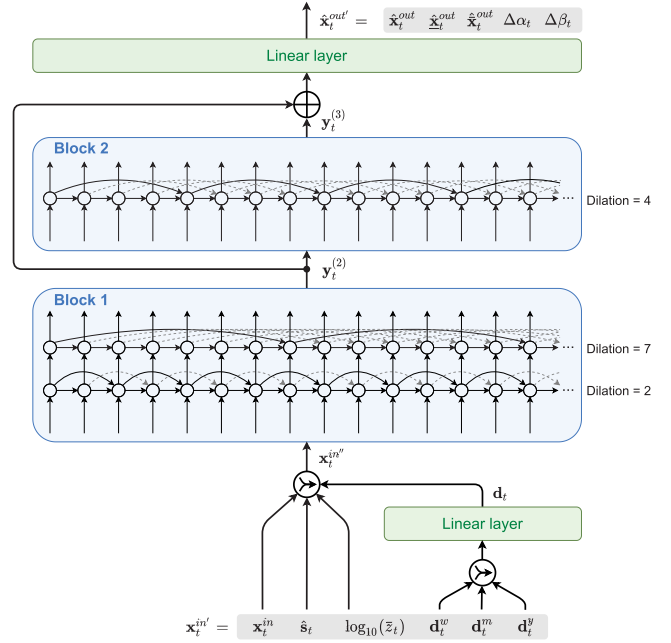


Fig. 8. Proposed RNN architecture. The circles represent dRNNCells.

was introduced in [36] to tackle the three major challenges presented by RNN when learning on long sequences, i.e., complex dependencies, vanishing and exploding gradients, and efficient parallelization. Note that our new dRNNCell is fed by both recent ($t - 1$) and delayed ($t - d$) states. Thanks to this, the cell uses direct information from both the previous step and a step distant in time. This can be useful for seasonal TS, where the relationships between the series elements have a cyclical character. These relationships can be modeled more accurately using dilated connections related to seasonality. To enable RNNs to learn the temporal dependencies of different scales, we use multiple dilated recurrent layers stacked with hierarchical dilations. The proposed RNN uses ResNet-style shortcuts between blocks [38] to improve the learning process by preventing vanishing or exploding gradients.

As can be seen from Fig. 8, binary vectors encoding calendar data, $\mathbf{d}_t^w$, $\mathbf{d}_t^m$, and $\mathbf{d}_t^y$, are embedded using a linear layer into d -dimensional continuous vectors $\mathbf{d}_t$. This reduces input dimensionality and meaningfully represents sparse binary vectors in the embedding space. The embedding is learned along with the model itself.

The output layer in Fig. 8 is a linear one. It produces vector $\hat{\mathbf{x}}_t^{\text{out}'}$ which is a concatenation of the forecast output pattern, $\hat{\mathbf{x}}_t^{\text{out}} = [\hat{x}_\tau]_{\tau \in \Delta_t^{\text{out}}}$, lower bounds of PI, $\hat{\underline{\mathbf{x}}}_t^{\text{out}} = [\hat{\underline{x}}_\tau]_{\tau \in \Delta_t^{\text{out}}}$, upper bounds of PI, $\hat{\bar{\mathbf{x}}}_t^{\text{out}} = [\hat{\bar{x}}_\tau]_{\tau \in \Delta_t^{\text{out}}}$, and corrections for smoothing coefficients, $\Delta\alpha_t$ and $\Delta\beta_t$

$$\hat{\mathbf{x}}_t^{\text{out}'} = [\hat{\mathbf{x}}_t^{\text{out}}, \hat{\underline{\mathbf{x}}}_t^{\text{out}}, \hat{\bar{\mathbf{x}}}_t^{\text{out}}, \Delta\alpha_t, \Delta\beta_t]. \quad (16)$$

D. Loss Function

To define the loss function we employ a pinball loss

$$\rho(z, \hat{z}_q) = \begin{cases} (z - \hat{z}_q)q, & \text{if } z \geq \hat{z}_q \\ (z - \hat{z}_q)(q - 1), & \text{if } z < \hat{z}_q \end{cases} \quad (17)$$

where z is an actual value, $\hat{z}_q$ is a forecast value of q th quantile, and $q \in (0, 1)$ is a quantile order.

The pinball loss is commonly used in quantile regression and probabilistic forecasting [39]. It helps us to determine the point forecasts and PIs, whose lower and upper bounds are expressed by quantiles of orders $\underline{q}$ and $\bar{q}$, respectively (e.g., $\underline{q} = 0.05$ and $\bar{q} = 0.95$).

Our loss function has the following three components:

$$L_\tau = \rho(z'_\tau, \hat{z}'_{q^*, \tau}) + \gamma(\rho(z'_\tau, \hat{z}'_{\underline{q}, \tau}) + \rho(z'_\tau, \hat{z}'_{\bar{q}, \tau})) \quad (18)$$

where $q^* = 0.5$ corresponds to the median, $z'_\tau = z_\tau / \bar{z}_t$ is a normalized actual TS value from the output window Δ_t^{out} , $\hat{z}'_{q^*, \tau} = \exp(\hat{x}_\tau) \hat{s}_{t, \tau}$ is a forecast value of z'_τ , $\underline{q}, \bar{q}$ are the quantile orders for the lower and upper bounds of PI, respectively, $\hat{z}'_{\underline{q}, \tau} = \exp(\hat{x}_\tau) \hat{s}_{t, \tau}$ is a forecast value of $\underline{q}$ -quantile of z'_τ , $\hat{z}'_{\bar{q}, \tau} = \exp(\hat{x}_\tau) \hat{s}_{t, \tau}$ is a forecast value of $\bar{q}$ -quantile of z'_τ , 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.

Note that loss function (18) operates on the normalized TS values z'_τ . This is because different TS can have different levels and normalization allows us to bring their errors expressed by (17) to the same level, which is crucial in cross-learning. The forecasts of z'_τ in (18) are calculated from (7) excluding $\bar{z}_t$ to obtain normalized forecasts. These forecasts are based on the RNN outputs, $\hat{\mathbf{x}}_t^{\text{out}}$, $\hat{\mathbf{x}}_t^{\text{out}}$ and $\hat{\mathbf{x}}_t^{\text{out}}$, and ES outputs $\{\hat{s}_{t, \tau}\}_{\tau \in \Delta_t^{\text{out}}}$.

The first component in (18), $\rho(z'_\tau, \hat{z}'_{q^*, \tau})$, represents a symmetrical loss for the forecast value (normalized) while the second and third components, $\rho(z'_\tau, \hat{z}'_{\underline{q}, \tau})$ and $\rho(z'_\tau, \hat{z}'_{\bar{q}, \tau})$, represent asymmetrical losses for the quantiles. The asymmetry level, which determines PI, results from the quantile orders. Hyperparameter γ determines the share of the three components in the loss function. For $\gamma = 1$, all the components have the same impact on the loss function. To increase the importance of the first component over the other two, we decrease the γ value. Note that due to the three-component parametrized loss function, we have the ability to optimize both the point forecasts and their PI. Moreover, the pinball loss allows us 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 [30], [31]).

E. Mechanisms and Solutions for Performance Improvement

The proposed model has the following mechanisms and solutions for improved performance.

- 1) *dRNNCell With Expanded States (Recent and Dilated Ones)*: dRNNCell is able to model both short-term and long-term dependencies in TS. This feature is useful, especially for STLTF where TS expresses multiple seasonality. Temporal dependencies in this case have a cyclical character and can be modeled hierarchically using different dilations in different RNN layers.
- 2) *Hybrid Architecture Combining ES and RNN*: ES extracts dynamically the main components of each individual TS and enables appropriate TS representation

for RNN. A multiple dilated stacked RNN architecture is able to deal with complex TS expressing multiple seasonality. The two components, ES and RNN, are optimized simultaneously by the same optimization algorithm. This fine-tunes RNN weights as well as ES smoothing coefficients. So the resulting forecasting model, including dynamic data preprocessing, is optimized as a whole.

- 3) *Cross-Learning*: The model is global. Learning across many TS enables it to capture the shared features and components of the TS. Cross-learning is a type of multitask learning [40] which is known to be an effective method of improving generalization by using the domain information contained in the training samples of related tasks as an inductive bias. Moreover, cross-learning greatly speeds up the learning of deep architectures.
- 4) *A Dynamic Training Set for RNN*: The training samples are updated on-the-fly during learning. The optimal representation of TS is searched for to ensure the best predictive performance of the model.
- 5) *Delaying the Moment of Over-training*: To delay the onset of the over-training, the starting point of training is sampled, so the same TS is likely to look slightly different each time a batch is formed.
- 6) *Three-Component, Parametrized Pinball Loss Function*: This enables the model to optimize both the point forecasts and their PIs. Moreover, it enables the forecast bias to be reduced.
- 7) *Ensembling*: Which is a powerful regularization technique. This exploits the beneficial effects of combining forecasts [41], improves accuracy and stability compared to a single learner, and enhances the robustness, thus mitigating model and parameter uncertainty [42].

IV. EXPERIMENTAL STUDY

In this section, we apply the proposed ES-dRNN model to STLTF and compare its performance with that of other models including statistical and ML ones. We test the models on real-world data comprising hourly electricity demand TS for 35 European countries from the period 2016–2018 (source—ENTSO-E repository www.entsoe.eu/data/power-stats/; we share this data with the ES-dRNN code in our GitHub repository [33]). The TS was described and analyzed in Section II. They differ substantially in levels, trends, dispersion, and daily shapes. Thus, the data provides a variety of TS with different properties, which translates into a more reliable test for the forecasting models.

A one-day-ahead forecasting problem is considered. We optimize ES-dRNN using the data from 2016 and 2017 (data for 2016 for Albania is unavailable). The model forecasts the daily load profile for each day of 2018 for each of the 35 countries with the exception of Estonia and Italy for which data for the last month of 2018 is unavailable, and Latvia for which data for the last two months of 2018 is unavailable.

The model was implemented in Python using PyTorch. It was run on an eight-core CPU (AMD Ryzen 7 1700, 3.0 GHz, 32 GB RAM).

A. Optimization and Training Procedures

During each epoch a number of updates are executed, guided by the average error accumulated by executing l_o (e.g., 50) forward steps, moving by one day, on a batch. The starting point is chosen randomly; the batches include random b series. The model is trained using Adam optimizer.

The d -dilated dRNNCell operates as described above only after d steps, because only after d steps are the delayed states available. In addition, the Holt-Winters formulas require at least twice the seasonality steps to stabilize, so the system uses several weeks (w_o) at the beginning of each batch as a warm-up period, during which all the ES and RNN calculations take place, with the exception of the training errors, which are not calculated. Similarly, an even longer warm-up period w_s is applied when producing the test results.

An epoch is usually defined as using all the training data once. Our definition here is based on the number of updates or processed batches, as during training we step l_o times on a batch (with random assignment of series) and for each batch execute a single update based on the average error. We aim to define the epoch as the number of updates that bring in a meaningful change in the learning process, and because the dataset contains a small number of series, a single epoch is actually composed of n_o number “subepochs,” defined in the traditional fashion as one scan of all available data. An additional factor is the batch size: when it grows, the number of updates per subepoch diminishes, so the number of subepochs needs to grow. However, in our experience, the linear growth is too fast, and risks overfitting within a single epoch, so finally we use the following formula:

$$n_o = \min \left(1, \left(\frac{Nb}{L} \right)^p \right) \quad (19)$$

where N is the maximum number of updates per epoch, b is the current batch size, L is the number of TS in the dataset, and p is a hyperparameter, between 0 and 1, which by experimentation is set to 0.7.

The pseudo-code for the ES-dRNN training algorithm is shown in Algorithm 1.

The model hyperparameters were selected as follows.

- 1) *Number of Epochs*: In early testing we established that 9 epochs is usually sufficient to reach a plateau of accuracy.
- 2) *Number of TS in the Batch*: We use the schedule of increasing batch sizes and decreasing learning rates proposed in [43]. We start with a small batch size of 2, and increase it, although only once, due to the small number of series, to 5 at epoch 4.
- 3) *Learning Rates*: Decreasing learning rates has a similar, if not the same, effect as increasing the batch size: it allows the validation error to be further reduced. We use the following schedule: $3 \cdot 10^{-3}$ (epochs 1–4), 10^{-3} (epoch 5), $3 \cdot 10^{-4}$ (epoch 6), and 10^{-4} (epochs 7–9).
- 4) *Size of the c-State, h-State, and y-Output*: $s_c = 100$, $s_h = 40$, $s_y = 60$. Increasing the size of cells causes a quadratic increase in the number of parameters. Larger models may be beneficial for larger datasets. The values

Algorithm 1 Pseudo-Code for the Training Algorithm

```

for  $iEpoch$  in range( $Num\_of\_Epochs$ ) do
  if Scheduled then
    Update learning rate and/or batch size
  end if
  Calculate number of sub-epochs
  for  $iSubEpoch$  in range( $Num\_of\_SubEpochs$ ) do
    Shuffle series and create batches
    for  $batch$  in batches do
      – Reset state of the RNN
      – Step through first Input_Window number of
        steps and update levels and seasonality, using the
        per-series  $\alpha_t$  and  $\beta_t$  (RNN not used)
      – Step through the remaining warm-up steps and
        training steps and update levels and seasonality,
        using the per-series  $\alpha_t$  and  $\beta_t$  adjusted by the RNN;
        if warm-up finished calculate the loss, average across
        all batch members, add the loss to per-batch list
      – Calculate average loss, update
    end for
  end for
end for

```

above were obtained by experimentation starting with $s_c = 50$, $s_h = 20$, and doubling it three times.

- 5) *RNN Architecture and Dilations*: We use three layers: two dilated by 2 and 7 in the first block, and a single layer dilated by 4 in the second block (see Fig. 8). We arrived at this layout partly by heuristics and partly by experimentation. We started with two layers, the first with a minimum dilation of 2 (because the previous values, dilated by 1, are always used by the cells), and the second layer dilated by 7, to match them with the main weekly seasonality. Then, we tried to add a third layer, and to avoid the vanishing gradient problem it had to be in a new block. We chose dilation 4 because together with 2 and 7, this makes an almost perfect geometric series, as advocated by [36]. But of course, we also experimented with larger dilation 11 and 14 in the third layer, but the results were worse, which is as expected because our horizon is just 1 day. Among the alternative arrangements for the three layers, three blocks of a single layer each dilated 2, 4, and 7 works equally well.
- 6) *Loss Function Parameters*: As described in Section III-D, the pinball loss function was utilized, with three different quantile values q , to achieve quantile regression for 0.5, 0.05, and 0.95. The actual values for q^* , $\underline{q}$, and $\bar{q}$ were slightly different: 0.49, 0.035, 0.96. These values were arrived at by experimentation, reducing the bias of the center value, and fine-tuning the percentage of exceedance for PIs. The γ parameter was 0.3, set, without experimentation, following a rule of thumb that states that the sum of PIs losses should be around 10%–20% of the center value loss, reflecting the usual higher importance applied to the center loss. However, it is not a sensitive parameter, and leaving it at 1 would not make much difference.

- 7) *Initial Smoothing Coefficients*: $I\alpha = -3.5$, $I\beta = 0.3$. These were arrived at by observing, during early runs of the training, the direction and size of average adjustments to both smoothing coefficients, by the NN. As expected, the level is quite stable, so the smoothing coefficient tends to be close to zero, and therefore the $I\alpha$ is a relatively large negative number. Seasonality, on the other hand, is likely to change more, and this is confirmed by larger, typically above 0.5, smoothing coefficients, starting from $I\beta = 0.3$.
- 8) *Lengths of TS Sequences in the Optimization Mode*: $l_o = 50$. The longer the sequence, the more smooth the gradient should be, but at the same time the system may “see” the same parts of a series too often and can overtrain quickly. In addition, experimentation suggested that smaller values of 20 and 30 brought worse results.
- 9) *Lengths of Training w_o and Testing w_s Warm-Up Periods*: Three and 13 weeks, respectively. The training warm-up period w_o needs to be just slightly longer than two weeks, twice the seasonality size, for ES to stabilize. The number of warming-up steps should also be larger than the smallest dilation, but this is just two steps (days), so this second condition is not important here. The length of the testing warm-up period was chosen mostly due to prior experience that stepping through two to three months of typical business TS is enough for the system to fully “zero-in” on a particular series.
- 10) *Embedding Size of the Calendar Variables*: This value was arrived at first by the expectation that the one-hot encoded input of size $7 + 31 + 52$ should be able to be converted to an order of magnitude smaller floating point vector, and then by experimentation.
- 11) *Ensemble size*: $E = 100$, although a size as small as 5 is often sufficient.

B. Baseline Models

We compare our ES-dRNN in terms of accuracy with the baseline models outlined below:

- 1) Naive—Naive model in the form: the forecast demand profile for day i is the same as the profile for day $i - 7$;
- 2) ARIMA—Autoregressive integrated moving average model [44];
- 3) ES—Exponential smoothing model [44];
- 4) Prophet—Modular additive regression model with nonlinear trend and seasonal components [11];
- 5) N-WE—Nadaraya–Watson estimator [44];
- 6) GRNN—General regression NN [15];
- 7) MLP—Perceptron with a single hidden layer and sigmoid nonlinearities [15];
- 8) SVM—Linear epsilon insensitive support vector machine (ϵ -SVM) [45];
- 9) LSTM—Long–short-term memory [46];
- 10) ANFIS—Adaptive neuro-fuzzy inference system [47];
- 11) MTGNN—Graph NN for multivariate TS forecasting [48];
- 12) DeepAR—Autoregressive RNN model for probabilistic forecasting [49];

TABLE I
FORECAST RESULTS

	MAPE	MdAPE	IqrAPE	RMSE	MPE	StdPE	SuccRate
Naive	5.08	4.84	3.32	704.34	-0.26	7.91	0
ARIMA	3.30	3.01	3.00	475.09	-0.01	5.31	0
ES	3.11	2.88	2.73	439.26	0.01	5.13	0
Prophet	4.53	4.32	3.03	619.39	-0.13	6.82	0
N-WE	2.49	2.28	2.30	332.49	-0.13	4.26	0
GRNN	2.48	2.28	2.27	332.91	-0.11	4.25	2.86
MLP	3.05	2.78	2.94	419.01	-0.04	5.07	0
SVM	2.55	2.29	2.52	357.24	-0.13	4.37	2.86
LSTM	2.76	2.57	2.52	381.76	0.02	4.47	0
ANFIS	3.65	3.17	3.66	507.08	-0.10	6.43	0
MTGNN	2.99	2.74	2.69	405.18	-0.47	4.85	0
DeepAR	3.42	3.25	2.95	487.14	-0.51	5.16	0
WaveNet	3.03	2.84	2.69	417.49	-0.83	4.68	0
N-BEATS	2.56	2.36	2.39	356.83	-0.04	4.29	5.71
LGBM	3.03	2.74	2.65	414.19	-0.25	4.68	0
XGB	2.96	2.69	2.62	405.91	0.01	4.61	0
ES-RNNe	2.38	2.18	2.26	324.11	-0.25	4.07	2.86
ES-dRNN	2.33	2.13	2.23	320.98	-0.20	3.89	0
ES-dRNNe	2.23	2.02	2.15	306.94	-0.20	3.75	85.71

- 13) WaveNet—Autoregressive deep-NN model combining causal filters with dilated convolutions [50];
- 14) N-BEATS—Deep NN with hierarchical doubly residual topology [51];
- 15) LGBM—Light gradient-boosting machine based on decision trees [52];
- 16) XGB—eXtreme gradient boosting algorithm based on decision trees [53]; and
- 17) ES-RNNe—Hybrid residual dilated LSTM and ES model [31].

The baseline models include classical statistical models (ARIMA and ES), new statistical models (Prophet), non-parametric pattern-based ML models (N-WE), classical ML models (MLP, GRNN, SVM, and ANFIS), new recurrent and deep-NN architectures (LSTM, MTGNN, DeepAR, WaveNet, and N-BEATS), boosted trees-based models (XGB and LGBM) and a predecessor of the proposed model (ES-RNNe).

C. Results

In this section, we report the results for our proposed model in two variants: as an individual model, denoted by ES-dRNN, and as an ensemble of E ES-dRNNs, denoted by ES-dRNNe.

Table I shows the results of forecasting averaged over all 35 countries, i.e., 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). MdAPE measures the average error without the influence of outliers, while RMSE is especially sensitive to outliers as a square error. MPE measures the forecast bias. Note the lowest values for MAPE, MdAPE, and RMSE for ES-dRNNe and the second lowest for ES-dRNN. Our models also produce the least dispersed predictions compared to the baseline models ($IqrAPE \leq 2.25$). The models that produce the least biased forecasts are XGB, ES, ARIMA, and LSTM. Our model is equipped with a bias reduction mechanism. However, excessive MPE reduction leads to an increase in the remaining error measures.

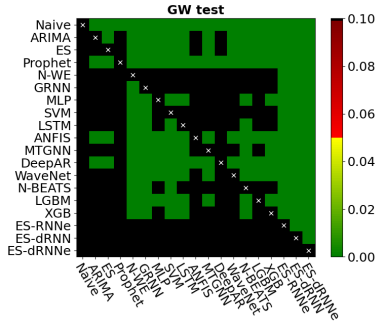


Fig. 9. Results of the GW tests.

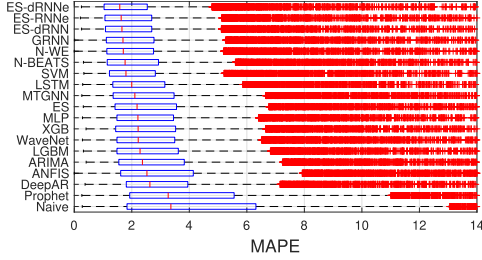


Fig. 10. Boxplots for daily MAPE.

The winning performance of ES-dRNNe and ES-dRNN was confirmed using a pairwise one-sided Giacomini-White test (GW test) for conditional predictive ability [54]. We used an implementation of the GW test in the multivariate variant from <https://github.com/jeslago/epftoolbox> [55]. Fig. 9 shows the results of the GW test, i.e., a heat map representing the obtained p -values. The closer they 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 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. Note that both ES-dRNNe and ES-dRNN performed significantly better in terms of accuracy than all the other comparative models.

More detailed results are shown in Figs. 10–12. From Fig. 10, we can assess the distribution of the daily MAPE. Note the smallest medians and the most compact distributions for our model, which is ahead of the group of nonparametric ML models designed specifically for STLF. Fig. 11 shows MAPE for individual countries. Our model was the most accurate for most of the countries apart from Estonia and Iceland, where it was beaten by N-BEATS, France, where it was beaten by SVM, Montenegro, where it was beaten by GRNN, and Czechia, where it was beaten by ES-RNNe. The model success rates based on the mean errors for the individual countries are shown in Table I (SuccRate).

Fig. 12 shows the average errors for each hour of the day, each day of the week, and each month of the test period (2018). It gives an overview of the accuracy of modeling individual seasonal components. Note that in each case errors for ES-dRNNe and ES-dRNN are among the lowest. The annual component is modeled most accurately in September and August, while the greatest errors occur in the spring months of March, April, and May. Note that our model is not very

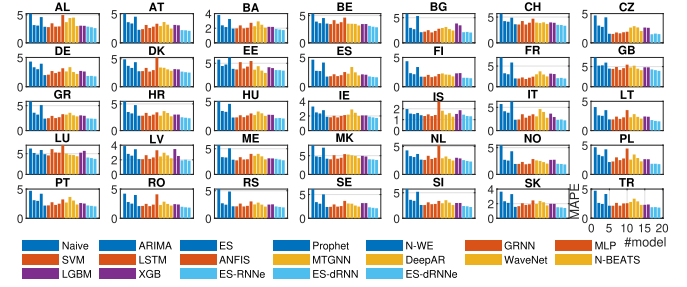


Fig. 11. MAPE of the models for each country.

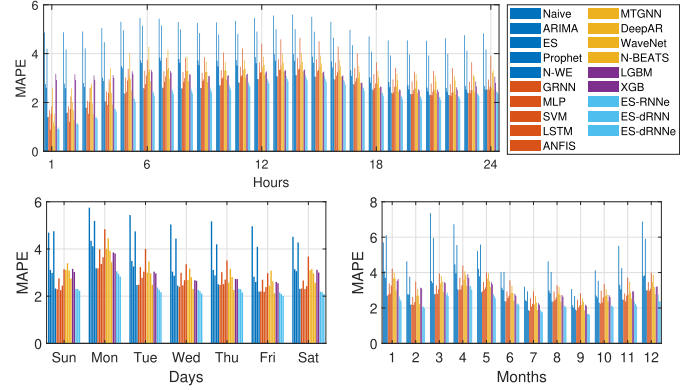


Fig. 12. MAPE for each hour of the day, each day of the week, and each month.

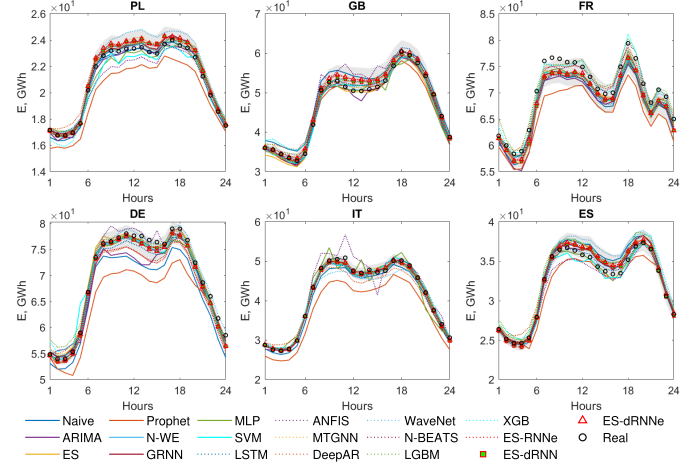


Fig. 13. Examples of the forecast daily profiles. 90% PIs for ES-dRNNe are shown as gray-shaded areas.

sensitive to the day of the week. The errors for individual days of the week are at a similar level, $\text{MAPE} \approx 2\% - 2.2\%$, except for Mondays where the errors reach 2.8%. Daily seasonality is modeled most accurately in the first hours of the day ($\text{MAPE} \approx 0.9\% - 1.3\%$), and least accurately in the period from 1 to 3 P.M., in which MAPE reaches 3%.

Fig. 13 shows some examples of the daily profile forecasts produced by our models and baseline models. From Fig. 13, we can assess the fitting of the models to the real data. In Fig. 13, the PIs predicted by ES-dRNNe are also shown. To assess the PIs, we calculate for each country the number of observed values in PIs, below PIs and above PIs. We achieved: $90.14\% \pm 2.43\%$, $4.88\% \pm 1.29\%$, and $4.98\% \pm 1.41\%$,

TABLE II
TRAINING TIME OF THE FORECASTING MODELS

Model	ARIMA	ES	Prophet	N-WE	GRNN	MLP	SVM	LSTM	ANFIS	MTGNN	DeepAR	WaveNet	N-BEATS	LGBM	XGB	ES-RNN	ES-dRNN
Time	180h	50h	7d	15s	8s	217m	27m	28h	18h	3h	42m	7h	32m	10m	40m	70m	46m

respectively. These values correspond to our assumed 90% PIs with lower and upper bounds $\underline{q} = 0.05$ and $\bar{q} = 0.95$, respectively.

Table II compares the computation time of the forecasting models for producing forecasts for each day of the test period for 35 countries. This is the total training time and query response time for each individual model. It does not include ensembling (training of many models) and optimization of the models except for ARIMA, ES, and Prophet whose implementations include automatic optimization. As can be seen from Table II, N-WE and GRNN models, which are lazy learners and do not require training, are the fastest. They are able to produce forecasts in seconds. The next fastest models are LGBM, SVM, and N-BEATS. The computation time for them was from 10 to 32 min. Our proposed model with a computation time of 46 min belongs to the third fastest group, which includes XGB and DeepAR. The computation time for the remaining models was over an hour.

D. Ablation Study

The proposed ES-dRNN has several components and mechanisms to increase its predictive power for STLTF. In the ablation study, we test the performance of the reduced model. We reduce the model as follows.

- Ab1** ES component is removed. dRNN learns on the normalized, but not deseasonalized TS. Seasonality vector $\hat{s}_t$ is excluded from input pattern (6).
- Ab2** ResNet-style shortcut between blocks 1 and 2 is removed. ES-dRNN learns without residual connection. The linear output layer is fed directly with the output vector of block 2, $\mathbf{y}_t^{(3)}$ [see Fig. (8)].
- Ab3** dRNNCell without fusion gate is used. The delayed c-state is used, if available, otherwise recent, $t - 1$, the state is used.
- Ab4** dRNNCell without dilated states is used. It is fed with only $t - 1$ states.
- Ab5** dRNNCell without recent states is used. It is fed with only $t - d$ states.
- Ab6** LSTM cell is used instead of dRNNCell. The cell is simplified, without delayed connections.
- Ab7** Level input $\log_{10}(\bar{z}_t)$ is excluded from input pattern (6).
- Ab8** Inputs $\log_{10}(\bar{z}_t)$, $\mathbf{d}_t^w$, $\mathbf{d}_t^m$, and $\mathbf{d}_t^y$ are excluded from input pattern (6). No input information about the TS level and current location in the weekly, monthly, and yearly cycles is introduced to dRNN.
- Ab9** Inputs $\log_{10}(\bar{z}_t)$, $\mathbf{d}_t^w$, $\mathbf{d}_t^m$, $\mathbf{d}_t^y$, and $\hat{s}_t$ are excluded from input pattern (6). dRNN is fed with only input

TABLE III
ERRORS FOR FULL AND REDUCED MODEL

	Full	Ab1	Ab2	Ab3	Ab4	Ab5	Ab6	Ab7	Ab8	Ab9	Ab10
MAPE	2.227	2.267	2.247	2.240	2.231	2.264	2.245	2.232	2.286	2.345	2.291
RMSE	306.9	310.2	310.5	309.1	307.0	310.9	307.2	307.9	313.1	321.8	315.4

pattern $\mathbf{x}_t^{\text{in}}$. Additional input information such as TS level, current seasonality, and calendar data is removed.

Ab10 Embedding is excluded. Extended input vector (6) is directly introduced on block 1. Input linear layer transforming calendar one-hot vectors into a continuous embedding vector is removed.

As can be seen from Table III, the lowest errors were achieved by the full model. Any reduction in the model leads to an increase in the forecast error. The most beneficial way of improving the accuracy of the model was extending the input pattern with daily variability, TS level, and calendar variables (Ab9). A model without this extension produced 5.3% worse forecasts in terms of MAPE. The second most beneficial procedure was embedding the calendar variables using a linear layer (Ab10). This resulted in a reduction in MAPE of 2.8%. Excluding the ES component from the model (Ab1) resulted in an increase in MAPE of 1.8% while excluding the recent states from dRNNCell (Ab5) resulted in an increase in MAPE of 1.7%. Ab2, Ab3, Ab4, and Ab6 resulted in a MAPE reduction of less than 1%. Among them, excluding dilated states from dRNNCell (Ab4) reduced the error the least, by 0.18%.

E. Discussion

The experimental study proves that both variants of ES-dRNN clearly outperform all other models in terms of accuracy. The wide range of STLTF problems on which we have tested the algorithms increases our confidence in this conclusion. The distinguishing feature of our model from other ML baseline models is that it produces both point forecasts and PIs with a specified probability coverage. Thus, the user gains additional information about the uncertainty of the prediction.

The proposed model is equipped with several mechanisms and solutions for performance improvement (see Section III-E). Many of them were tested and proved their effectiveness in forecasting problems from diverse domains (see the winning submission to the M4 competition [30] and the model for monthly electricity demand forecasting [31]). Other components and mechanisms were designed in this study especially for STLTF to deal with complex seasonality and short and long-term dependencies. As the ablation study has shown, the most important of them turned out to be: extended input information including daily variability, TS level and the calendar variables (Ab9), embedding of the calendar data (Ab10), and ES component (Ab1). Removing these components and mechanisms worsens the results the most.

We confirmed the beneficial effect of ensembling on increasing the accuracy of the forecasting model. The errors for the ensemble version were lower than those for the individual version by 5.51% for MAPE, 6.05% for MdAPE, and 5.86%

for RMSE. In our approach, ensembling does not require additional effort related to the selection of additional hyperparameters, e.g., controlling the diversity of individual learners. The diversity of learners is provided by the random initialization of the model parameters. However, controlling diversity could be an additional way of improving performance. The individual ensemble members produced forecasts with MAPE ranging from 2.30% to 2.36%. MAPE standard deviation was 0.0145% and its ratio to mean error was 0.62%.

The proposed ES-dRNN is more complex than the baseline statistical and ML models. It has a larger number of parameters (around 229 K) and hyperparameters to tune. However, the development of the model did not require long processing times nor any special hardware—it was done on a desktop-class computer, without GPU. A single training takes less than 1 h and can be done in parallel using a number of workers, allowing the results of the ensemble to be calculated immediately. Our experience with similar models allowed us to limit the number of hyperparameter combinations and code modifications. It is also worth noting that, once this kind of model is trained, the NN weights can be saved, and a serving program that uses them can be built. Such a serving program can forecast automatically and rapidly (in a matter of seconds) all the TS when fed with new data. We worked with a relatively small dataset covering two to three years. Considering strong yearly seasonality, extensive tuning of the model, e.g., in order to completely remove bias, would likely have led to overfitting, so we purposefully avoided it.

V. CONCLUSION

In this article, we proposed and empirically validated a new hybrid hierarchical architecture for STLF—ES-dRNN. The empirical study of STLF for 35 European countries showed that our ES-dRNN had a significantly better performance than statistical and ML methods. It clearly outperformed its competitors in terms of accuracy. Its success is due to its unique hybrid architecture which combines ES and RNN. To deal with multiple seasonalities and short and long-term dependencies in TS we designed a new dilated recurrent cell and multiple dilated stacked RNN architecture. ES extracts dynamically the main components of each individual TS and enables appropriate TS representation for RNN. Due to the simultaneous learning of both ES and dRNN components, the model is optimized as a whole. Cross-learning, i.e., learning on multiple TS, enables ES-dRNN to capture the shared features and components of each individual TS.

The model produces point forecasts and PIs to express the forecast uncertainty. To optimize both, we introduced a new three-component, parametrized loss function. This also allows the forecast bias to be controlled. A major advantage of ES-dRNN is its ability to deal with raw TS without any kind of preprocessing such as decomposition or rationalization. All necessary data processing takes place inside the model.

The high expressive power of the proposed model to solve nonlinear stochastic forecasting problems with complex seasonalities and significant random fluctuations has encouraged us to apply it to solve other complicated forecasting problems. This will be the focus of our future work.

REFERENCES

- [1] G. Dudek, "Multilayer perceptron for short-term load forecasting: From global to local approach," *Neural Comput. Appl.*, vol. 32, no. 8, pp. 3695–3707, Apr. 2020.
- [2] B. F. Hobbs, S. Jitrapakulsarn, S. Konda, V. Chankong, K. A. Loparo, and D. J. Maratukulam, "Analysis of the value for unit commitment of improved load forecasts," *IEEE Trans. Power Syst.*, vol. 14, no. 4, pp. 1342–1348, Mar. 1999.
- [3] S. Arora and J. W. Taylor, "Rule-based autoregressive moving average models for forecasting load on special days: A case study for France," *Eur. J. Oper. Res.*, vol. 266, pp. 259–268, Jan. 2018.
- [4] J. W. Taylor, "Short-term load forecasting with exponentially weighted methods," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 458–464, Feb. 2012.
- [5] N. Charlton and C. Singleton, "A refined parametric model for short term load forecasting," *Int. J. Forecasting*, vol. 30, no. 2, pp. 364–368, Apr. 2014.
- [6] H. Takeda, Y. Tamura, and S. Sato, "Using the ensemble Kalman filter for electricity load forecasting and analysis," *Energy*, vol. 104, pp. 184–198, May 2016.
- [7] S. Sharma, A. Majumdar, V. Elvira, and É. Chouzenoux, "Blind Kalman filtering for short-term load forecasting," *IEEE Trans. Power Syst.*, vol. 35, no. 6, pp. 4916–4919, Nov. 2020.
- [8] G. Dudek, "Pattern-based local linear regression models for short-term load forecasting," *Electr. Power Syst. Res.*, vol. 130, pp. 139–147, Jan. 2016.
- [9] J. W. Taylor, "Triple seasonal methods for short-term load forecasting," *Eur. J. Oper. Res.*, vol. 204, pp. 139–152, Jan. 2010.
- [10] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *J. Amer. Stat. Assoc.*, vol. 106, no. 496, pp. 1513–1527, Dec. 2011.
- [11] S. J. Taylor and B. Letham, "Forecasting at scale," *Amer. Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
- [12] S. Fan and R. J. Hyndman, "Short-term load forecasting based on a semi-parametric additive model," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 134–141, Feb. 2012.
- [13] B. A. Høverstad, A. Tidemann, H. Langseth, and P. Öztürk, "Short-term load forecasting with seasonal decomposition using evolution for parameter tuning," *IEEE Trans. Smart Grid*, vol. 6, no. 4, pp. 1904–1913, May 2015.
- [14] K. Benidis et al., "Deep learning for time series forecasting: Tutorial and literature survey," 2020, *arXiv:2004.10240*.
- [15] G. Dudek, "Neural networks for pattern-based short-term load forecasting: A comparative study," *Neurocomputing*, vol. 205, pp. 64–74, Sep. 2016.
- [16] H. S. Hippert and J. W. Taylor, "An evaluation of Bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting," *Neural Netw.*, vol. 23, no. 3, pp. 386–395, Apr. 2010.
- [17] Z. A. Bashir and M. E. El-Hawary, "Applying wavelets to short-term load forecasting using PSO-based neural networks," *IEEE Trans. Power Syst.*, vol. 24, no. 1, pp. 20–27, Feb. 2009.
- [18] G. Dudek, "Randomized neural networks for forecasting time series with multiple seasonality," in *Proc. 16th Int. Work-Confer. Artif. Neural Netw.* Cham, Switzerland: Springer, 2021, pp. 196–207.
- [19] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, and J. He, "Short-term load forecasting with deep residual networks," *IEEE Trans. Smart Grid*, vol. 10, no. 4, pp. 3943–3952, Jul. 2019.
- [20] H. J. Sadaei, P. C. de Lima e Silva, F. G. Guimarães, and M. H. Lee, "Short-term load forecasting by using a combined method of convolutional neural networks and fuzzy time series," *Energy*, vol. 175, pp. 365–377, May 2019.
- [21] X. Kong, C. Li, F. Zheng, and C. Wang, "Improved deep belief network for short-term load forecasting considering demand-side management," *IEEE Trans. Power Syst.*, vol. 35, no. 2, pp. 1531–1538, Mar. 2020.
- [22] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, Jan. 2019.
- [23] S. Wang, X. Wang, S. Wang, and D. Wang, "Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting," *Int. J. Electr. Power Energy Syst.*, vol. 109, pp. 470–479, Jul. 2019.
- [24] G. Brown, J. L. Wyatt, and P. Tiño, "Managing diversity in regression ensembles," *J. Mach. Learn. Res.*, vol. 6, pp. 1621–1650, Dec. 2005.

- [25] M. El-Hendawi and Z. Wang, "An ensemble method of full wavelet packet transform and neural network for short term electrical load forecasting," *Electr. Power Syst. Res.*, vol. 182, May 2020, Art. no. 106265.
- [26] D. Yang, J.-E. Guo, S. Sun, J. Han, and S. Wang, "An interval decomposition-ensemble approach with data-characteristic-driven reconstruction for short-term load forecasting," *Appl. Energy*, vol. 306, Jan. 2022, Art. no. 117992.
- [27] Y. Hu et al., "Short-term load forecasting using multimodal evolutionary algorithm and random vector functional link network based ensemble learning," *Appl. Energy*, vol. 285, Mar. 2021, Art. no. 116415.
- [28] Y. Wang et al., "Short-term load forecasting for industrial customers based on TCN-LightGBM," *IEEE Trans. Power Syst.*, vol. 36, no. 3, pp. 1984–1997, May 2021.
- [29] C. S. Lai et al., "Multi-view neural network ensemble for short and mid-term load forecasting," *IEEE Trans. Power Syst.*, vol. 36, no. 4, pp. 2992–3003, Jul. 2021.
- [30] 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.
- [31] G. Dudek, P. Pelka, and S. Smyl, "A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2879–2891, Jul. 2022.
- [32] 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.
- [33] *ES-dRNN Code and Data*. [Online]. Available: <https://github.com/slaweks17/ES-dRNN>
- [34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [35] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
- [36] S. Chang et al., "Dilated recurrent neural networks," in *Proc. NIPS*, 2017, pp. 1–15.
- [37] I. Ben-Ari and R. Shwartz-Ziv, "Sequence modeling using a memory controller extension for LSTM," in *Proc. NIPS*, 2017, pp. 1–12.
- [38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [39] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile estimation," *J. Mach. Learn. Res.*, vol. 7, pp. 1231–1264, Jul. 2006.
- [40] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, Jul. 1997.
- [41] F. Chan and L. L. Pauwels, "Some theoretical results on forecast combinations," *Int. J. Forecasting*, vol. 34, no. 1, pp. 64–74, Jan. 2018.
- [42] 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.
- [43] S. L. Smith, P. J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *Proc. ICLR*, 2018, pp. 1–10.
- [44] G. Dudek, "Pattern similarity-based methods for short-term load forecasting—Part 2: Models," *Appl. Soft Comput.*, vol. 36, pp. 422–441, Nov. 2015.
- [45] P. Pelka, "Pattern-based forecasting of monthly electricity demand using support vector machine," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8, doi: [10.1109/IJCNN52387.2021.9534134](https://doi.org/10.1109/IJCNN52387.2021.9534134).
- [46] P. Pelka and G. Dudek, "Pattern-based long short-term memory for mid-term electrical load forecasting," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.
- [47] P. Pelka and G. Dudek, "Neuro-fuzzy system for medium-term electric energy demand forecasting," in *Proc. 38th Int. Conf. Inf. Syst. Archit. Technol. (ISAT)*. Cham, Switzerland: Springer, 2018, pp. 38–47.
- [48] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 1–6.
- [49] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *Int. J. Forecasting*, vol. 36, no. 3, pp. 1181–1191, Jul. 2020.
- [50] A. van den Oord et al., "WaveNet: A generative model for raw audio," 2016, *arXiv:1609.03499*.
- [51] B. N. Oreshkin, D. Carpio, N. Chapados, and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–17.
- [52] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–5.
- [53] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1–10.
- [54] R. Giacomini and H. White, "Tests of conditional predictive ability," *Econometrica*, vol. 74, no. 6, pp. 1545–1578, 2006.
- [55] J. Lago, G. Marcjasz, B. De Schutter, and R. Weron, "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark," *Appl. Energy*, vol. 293, Jul. 2021, Art. no. 116983.



Slawek Smyl received the M.Sc. degree in physics from Jagiellonian University, Kraków, Poland, in 1988, the M.Eng. degree in information technology from RMIT, Melbourne, VIC, Australia, in 1997, and the Grad.D. degree in legal studies from UNSW, Sydney, NSW, Australia, in 2004.

He is currently a Quantitative Engineer at Facebook, Menlo Park, CA, USA, working in the area of time series forecasting.

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



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

Currently, he is an Associate Professor at the Department of Electrical Engineering, CUT. 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 came third at the Global Energy Forecasting Competition in 2014 (price forecasting track).



Pawel Pelka received the Ph.D. degree in informatics from the Częstochowa University of Technology, Częstochowa, Poland, in 2021. His Ph.D. thesis was focused on pattern-based mid-term load forecasting issues.

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

ES-dRNN with Dynamic Attention for Short-Term Load Forecasting

Slawek Smyl

Meta

1 Hacker Way, Menlo Park, CA 94025, USA
slawek.smyl@gmail.com

Grzegorz Dudek

Department of Electrical Engineering

Czestochowa University of Technology

Czestochowa, Poland

grzegorz.dudek@pcz.pl

Paweł Pełka

Department of Electrical Engineering

Czestochowa University of Technology

Czestochowa, Poland

pawel.pelka@pcz.pl

Abstract—Short-term load forecasting (STLF) is a challenging problem due to the complex nature of the time series expressing multiple seasonality and varying variance. This paper proposes an extension of a hybrid forecasting model combining exponential smoothing and dilated recurrent neural network (ES-dRNN) with a mechanism for dynamic attention. We propose a new gated recurrent cell – attentive dilated recurrent cell, which implements an attention mechanism for dynamic weighting of input vector components. The most relevant components are assigned greater weights, which are subsequently dynamically fine-tuned. This attention mechanism helps the model to select input information and, along with other mechanisms implemented in ES-dRNN, such as adaptive time series processing, cross-learning, and multiple dilation, leads to a significant improvement in accuracy when compared to well-established statistical and state-of-the-art machine learning forecasting models. This was confirmed in the extensive experimental study concerning STLF for 35 European countries.

Index Terms—exponential smoothing, hybrid forecasting models, multiple seasonality, recurrent neural networks, short-term load forecasting, time series forecasting

I. INTRODUCTION

STLF is a challenging problem due to the complex nature of the time series. Electricity load or demand time series exhibit three seasonal components (annual, weekly and daily), a nonlinear trend, varying variance and random fluctuations. The daily load pattern differs in shape depending on the day of the week and season of the year. The load time series is strongly influenced by climatic, weather and economic conditions, all of which have a stochastic nature. All these properties place high demands on forecasting models. STLF is considered to be a very challenging problem, which is why it is often used for testing new models.

STLF is also very important in practice as an integral part of power system control and scheduling. It is needed for the efficient and safe operation of power systems and supporting transactions of participants in deregulated electricity markets. Due to the importance and complexity of the STLF problem, a large number of different STLF models have been reported in the literature. They employ conventional statistical methods, computational intelligence and machine learning methods as

well as hybrid solutions. The most popular new approaches for STLF are based on neural networks (NNs) [1]. They offer more possibilities than statistical models and exceed many of their limitations. These limitations include their linear nature, limited ability to model complicated seasonal patterns, limited adaptability, a shortage of expressive power, and problems with capturing long-term dependencies and introducing exogenous variables.

Due to their flexibility and universal approximation property, NNs can model any nonlinear relationship and reflect process variability in an uncertain dynamic environment. Thus, they are often used for complex forecasting problems such as STLF. The classical NN architectures such as multilayer perceptron (MLP), radial basis function (RBF) NN, generalized regression neural network (GRNN), fuzzy counterpropagation NN, and self-organizing maps were compared as STLF models in [2]. Multiple seasonality, which is a real problem for the forecasting models that usually requires decomposition or deseasonalisation, was solved in this study by appropriate representation of the time series defining patterns of the daily profiles. Thanks to this, the forecasting problem was simplified and it was possible to apply less complex neural models with a smaller number of parameters, which were more resistant to overfitting. Recently, classical MLP with pattern representation was replaced by randomized NN [3]. Due to randomized learning, the training becomes much faster and easier and the numbers of parameters and hyperparameters to adjust is significantly reduced. At the same time, the accuracy of STLF improves in relation to MLP. Many STLF approaches combine neural models with effective optimization procedure and time series decomposition or a feature engineering method. For example, in [4], the time series is decomposed into orthonormal series generated by a wavelet transform, and MLP is learned using a particle swarm optimization algorithm. In [5], to control MLP complexity and to select input variables, a Bayesian approach was used. The Bayesian framework offered ways to avoid overfitting by regularisation, to decide on the number of neurons and to deal with the inputs by soft-pruning.

The rapid development of deep and recurrent NNs in recent years has led to the development of effective STLF methods. The new possibilities they offer, which are very attractive for forecasting models, include learning of representation, cross-

Partially supported by grant 020/RID/2018/19 from the Polish Minister of Science and Higher Education titled "Regional Initiative of Excellence", 2019-22 (PP, GD).

learning on massive datasets and modeling temporary relationships in sequential data. Some examples of STLTF models based on deep learning are: [6], where deep residual NNs were proposed and applied to probabilistic load forecasting using Monte Carlo dropout; [7], where a multivariate fuzzy time series was converted into multi-channel images and processed by CNN to produce load forecasts; and [8], where an improved deep belief network for STLTF with demand-side management was proposed.

Recurrent NNs, which were designed for sequential data such as time series and text data are extremely useful for forecasting problems. Modern RNNs such as long-short term memory (LSTM) and gated recurrent unit (GRU) are distinguished by their ability to model both short and long-term dependencies in time series [9]. Examples of RNN for STLTF can be found in: [10], where a model based on bi-directional LSTM and attention mechanism was proposed; [11], where a STLTF problem for individual residential households was addressed using LSTM; and [12], where the load is decomposed into different frequency components using the empirical-mode decomposition algorithm and then low-frequency components are predicted using linear regression while the high-frequency components are predicted using LSTM.

In our recent work [13], motivated by new achievements in deep learning and RNNs [14], we proposed a hybrid hierarchical STLTF model combining exponential smoothing (ES) and dilated RNN (ES-dRNN). Its hybrid architecture improved ability to learn representation and explore hidden patterns. To deal with multiple seasonalities and temporal dependencies in time series, we proposed a new dilated recurrent cell (dRNNCell) and multiple dilated stacked RNN architecture. The simultaneous learning of both ES and dRNN components enables the entire model to be optimized while learning on multiple time series (cross-learning) enables ES-dRNN to capture the shared features of individual series. The model proposed in this work inherits ES-dRNN properties and extends it with a dynamic attention mechanism [15]. This mechanism allows the model to focus on relevant input information while the target functions for both point forecasts and predictive intervals are being modeled.

The contribution of this study includes the following points:

- 1) We propose a new attentive dilated recurrent cell, adRNNCell, which implements an attention mechanism for weighting the input information. It produces an internal attention vector which dynamically weights input vector components. This attention mechanism permits the recurrent cell to utilize the most relevant components of the input patterns in a flexible manner to improve the forecasting performance of the model.
- 2) We develop a hybrid exponential smoothing and dilated recurrent NN model with attention based on adRNNCell. The model produces vectors of point forecasts and also predictive intervals. Due to its internal mechanisms such as dynamic attention, adaptive time series processing, cross-learning, and multiple dilations, the model can deal with complex time series expressing multiple sea-

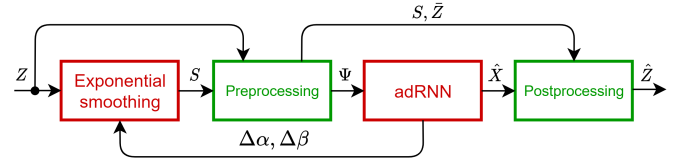


Fig. 1. Block diagram of the proposed ES-adRNN forecasting model.

sonality and varying variance. Our model is available as open-source code in the github repository [16].

- 3) We empirically demonstrate on real data sets of 35 European countries that our proposed model significantly outperforms in terms of STLTF accuracy its predecessor [13] and 13 baseline forecasting models, including well-established statistical approaches and state-of-the-art machine learning approaches.

The rest of the work is organized as follows. In Section II, we present the architecture of the hybrid exponential smoothing and dilated recurrent NN model with dynamic attention (ES-adRNN). Details of the proposed adRNNCell are described in Section III. The experimental framework used to evaluate the performance of the proposed approach is described in Section IV. Finally, Section V concludes the work.

II. ES-ADRNN ARCHITECTURE

A hybrid exponential smoothing and dilated recurrent NN model with an attention mechanism, ES-adRNN, is composed of four main components as shown in Fig. 1. This architecture is similar to that of ES-dRNN proposed in [13], except for the RNN component. In this study, we introduce RNN with a new gated recurrent cell, which is equipped with an attention mechanism. This new RNN component, adRNN, is described in details in Section III.

The proposed model works in a recursive manner, generating daily pattern forecasts for the following days in the subsequent recursive steps t . The model input, Z , represents a set of L time series: $\{\{z_\tau^l\}_{\tau=1}^{M_l}\}_{l=1}^L$, where M_l is an l -th time series length. In our case, the series express hourly electricity demand for L countries. The model learns simultaneously on L time series, i.e. in a cross-learning mode [14], which enables it to capture the shared features of the individual time series.

ES decomposes each series into level and seasonal components. The seasonal components, S , are used by a preprocessing component to deseasonalize the original series. The series are also normalized and squashed to prepare the training set, Ψ , for adRNN. adRNN learns not only the main mapping function transforming inputs into electricity demands for the next day (preprocessed) but also their predictive intervals (PIs) and the corrections of ES smoothing parameters, $\Delta\alpha$ and $\Delta\beta$. A postprocessing component transforms forecasts of the preprocessed demand and PIs, $\hat{X}$, into real values, $\hat{Z}$. It uses for this seasonal components, S , and average values of input sequences, $\bar{Z}$, determined by the ES and preprocessing components.

A. Exponential Smoothing

We use a simplified Holt-Winters multiplicative ES model, which decomposes the time series into two components: level component and seasonal component. A unique feature of the proposed approach is that the smoothing coefficients of the Holt-Winters model are not fixed, as in the typical case, but they are adapted in each recursive step t as follows:

$$\begin{aligned}\alpha_{t+1} &= \sigma(I\alpha + \Delta\alpha_t) \\ \beta_{t+1} &= \sigma(I\beta + \Delta\beta_t)\end{aligned}\quad (1)$$

where $\alpha, \beta \in [0, 1]$ are smoothing coefficients, $I\alpha$ and $I\beta$ are initial values of the smoothing coefficients, $\Delta\alpha_t$ and $\Delta\beta_t$ are the corrections, and σ denotes a sigmoid function.

The corrections are learned by adRNN simultaneously with the main mapping function. Thus, they have a dynamic character. They depend on the adRNN input representing the current time series characteristic and calendar variables. The dynamic version of the Holt-Winters multiplicative model takes the form:

$$\begin{aligned}l_{t,\tau} &= \alpha_t \frac{z_\tau}{s_{t,\tau}} + (1 - \alpha_t)l_{t,\tau-1} \\ s_{t,\tau+168} &= \beta_t \frac{z_\tau}{l_{t,\tau}} + (1 - \beta_t)s_{t,\tau}\end{aligned}\quad (2)$$

where $\{z_\tau\}_{\tau=1}^M$ is a decomposed time series, $l_{t,\tau}$ denotes a level component and $s_{t,\tau}$ denotes a weekly seasonal component.

B. Preprocessing and Postprocessing

The preprocessing component prepares training patterns for adRNN. An input pattern represents a weekly period covered by moving window Δ^{in} of size 168 hours. An output pattern represents the forecasted daily sequence covered by moving window Δ^{out} of size 24 hours, which follows Δ^{in} . The windows are shifted by 24 hours to obtain subsequent input and output patterns:

$$\begin{aligned}\mathbf{x}_1^{in} &= [x_1, \dots, x_{168}], & \mathbf{x}_1^{out} &= [x_{169}, \dots, x_{192}], \\ \mathbf{x}_2^{in} &= [x_{25}, \dots, x_{192}], & \mathbf{x}_2^{out} &= [x_{193}, \dots, x_{216}], \\ & \dots\end{aligned}\quad (3)$$

The components of the t -th pair of patterns express de-seasonalized, normalized and squashed time series sequences covered by the t -th pair of windows. They are determined as follows:

$$x_\tau = \log \frac{z_\tau}{\bar{z}_t \hat{s}_{t,\tau}} \quad (4)$$

where $\tau \in \Delta_t^{in} \cup \Delta_t^{out}$, $\bar{z}_t$ is the average value in Δ_t^{in} and $\hat{s}_{t,\tau}$ is the seasonal component predicted by ES (2) for recursive step t .

The log function in (4) squashes the data to prevent outliers from adversely affecting model performance. Note that the seasonal component in (4) is adapted in each recursive step t . This makes the training data dynamic and enables the model to learn data representation.

Input patterns $\mathbf{x}_t^{in}$ are the main component of the input vectors for adRNN learning. Additionally, to introduce more information related to the forecasted period, the input vectors include: (i) a seasonal vector predicted by ES for the t -th output period reduced by 1, i.e. $\hat{s}_t = [\hat{s}_{t,\tau-1}]_{\tau=24(t-1)+169}^{24(t-1)+192}$, (ii) a current level of the time series, $\log_{10}(\bar{z}_t)$, and (iii) calendar variables, $\mathbf{d}_t^w \in \{0, 1\}^7$, $\mathbf{d}_t^m \in \{0, 1\}^{31}$ and $\mathbf{d}_t^y \in \{0, 1\}^{52}$, as binary one-hot vectors encoding day of the week, day of the month and week of the year for the forecasted day, respectively. The input vector takes the form:

$$\mathbf{x}_t^{in'} = [\mathbf{x}_t^{in}, \hat{s}_t, \log_{10}(\bar{z}_t), \mathbf{d}_t^w, \mathbf{d}_t^m, \mathbf{d}_t^y] \quad (5)$$

The daily patterns forecasted by adRNN, $\hat{\mathbf{x}}_t^{out} = [\hat{x}_\tau]_{\tau \in \Delta_t^{out}}$, are transformed by the postprocessing component to obtain real forecasts of the hourly electricity demand. For this, transformed equation (4) is used:

$$\hat{z}_\tau = \exp(\hat{x}_\tau) \bar{z}_t \hat{s}_{t,\tau} \quad (6)$$

where $\tau \in \Delta_t^{out}$.

C. Loss Function

The proposed model predicts hourly electricity demands, z_τ , and their PIs in the form of two quantiles of orders q (lower) and $\bar{q}$ (upper). To optimize both point forecasts and PIs, we use a loss function in the form [13]:

$$L_\tau = \rho(z'_\tau, \hat{z}'_{q^*,\tau}) + \gamma[\rho(z'_\tau, \hat{z}'_{q,\tau}) + \rho(z'_\tau, \hat{z}'_{\bar{q},\tau})] \quad (7)$$

where ρ denotes a pinball loss function:

$$\rho(z, \hat{z}_q) = \begin{cases} (z - \hat{z}_q)q & \text{if } z \geq \hat{z}_q \\ (z - \hat{z}_q)(q - 1) & \text{if } z < \hat{z}_q \end{cases} \quad (8)$$

z is an actual value; $\hat{z}_q$ is a forecasted value of q -th quantile; $q \in (0, 1)$ is a quantile order; $q^* = 0.5$ corresponds to the median; $z'_\tau = z_\tau / \bar{z}_t$ is a normalized actual time series value from the output window Δ_t^{out} ; $\hat{z}'_{q^*,\tau} = \exp(\hat{x}_\tau) \hat{s}_{t,\tau}$ is a forecasted value of z'_τ ; q and $\bar{q}$ are the quantile orders for the lower and upper bounds of PI, respectively; $\hat{z}'_{q,\tau} = \exp(\hat{x}_\tau) \hat{s}_{t,\tau}$ is a forecasted value of q -quantile of z'_τ ; $\hat{z}'_{\bar{q},\tau} = \exp(\hat{x}_\tau) \hat{s}_{t,\tau}$ is a forecasted value of $\bar{q}$ -quantile of z'_τ ; and $\gamma \geq 0$ is a controlling parameter.

Loss function (7) uses normalized values of the time series, z'_τ , to bring the errors calculated for different time series to the same level, as this is important in cross-learning.

The first component in (7), $\rho(z'_\tau, \hat{z}'_{q^*,\tau})$, is almost (see below) a symmetrical loss for the point forecast (normalized) while the second and third components, $\rho(z'_\tau, \hat{z}'_{q,\tau})$ and $\rho(z'_\tau, \hat{z}'_{\bar{q},\tau})$, are asymmetrical losses for the quantiles. The asymmetry level, which determines PI, results from the quantile orders. Parameter γ enables us to control the impact of PI-related components on the loss value. Its typical value ranges from 0.1 to 0.5.

It is worth noting that the pinball loss gives us the opportunity to reduce the forecast bias by penalizing positive and

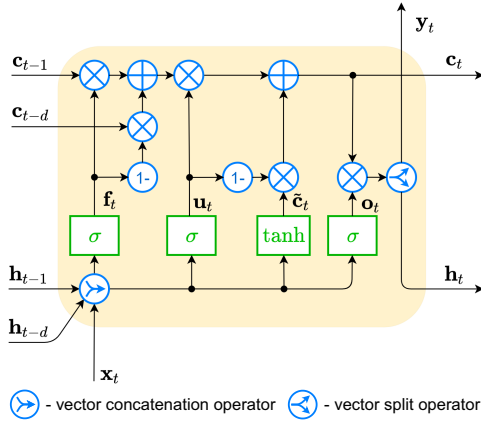


Fig. 2. Dilated recurrent cell dRNNCell.

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 [14], [17]).

III. DILATED RNN WITH DYNAMIC ATTENTION

The RNN component of ES-adRNN employs a dilated RNN cell (dRNNCell) introduced in [13]. This cell is shown in Fig. 2. It was designed for STLf to deal with a complex seasonal pattern expressing three seasonalities. The distinguishing features of dRNNCell are:

- dRNNCell is fed by two cell states (c -states) and two controlling states (h -states). They represent recent states, $\mathbf{c}_{t-1}$, $\mathbf{h}_{t-1}$, and delayed states, $\mathbf{c}_{t-d}$, $\mathbf{h}_{t-d}$, $d > 1$.
- The output of dRNNCell 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.

In this study, we combine two dRNNCells to obtain a more efficient gated recurrent cell, which is able to preprocess dynamically the input data. A new cell, attentive dilated RNN cell (adRNNCell), introduces an attention mechanism for weighting the input information. It is shown in Fig. 3. The first cell (at the bottom of the figure) produces attention vector $\mathbf{m}_t$ of the same length as 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 vector $\mathbf{x}_t$. Thus, they strengthen or weaken the particular inputs to the second cell (at the top of the figure). Note that $\mathbf{m}_t$ has a dynamical character - the weights are adjusted to the current inputs at time t . Both cells learn simultaneously. Based on preprocessed input vector, $\mathbf{x}_t^2$, the second cell produces vector $\mathbf{y}_t$, which feeds the next layer.

dRNNCell as a separate cell and as a component cell of adRNNCell, uses three gates. Namely, a fusion gate f , update gate u , and output gate o . The gates transform the input vectors, $\mathbf{x}_t$, $\mathbf{h}_{t-1}$ and $\mathbf{h}_{t-d}$, using sigmoid nonlinearity σ . A candidate c -state, $\tilde{\mathbf{c}}_t$, is produced by transforming input vectors using $\tanh$ nonlinearity. The dRNNCell formulation is as follows:

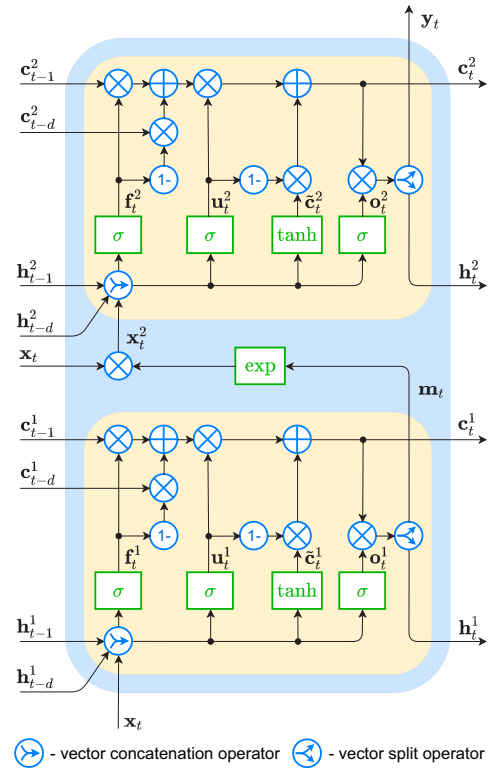


Fig. 3. Attentive dilated recurrent cell adRNNCell.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{V}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{h}_{t-d} + \mathbf{b}_f) \quad (9)$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_u \mathbf{x}_t + \mathbf{V}_u \mathbf{h}_{t-1} + \mathbf{U}_u \mathbf{h}_{t-d} + \mathbf{b}_u) \quad (10)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{V}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{h}_{t-d} + \mathbf{b}_o) \quad (11)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{V}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{h}_{t-d} + \mathbf{b}_c) \quad (12)$$

where subscript t denotes a time step; σ denotes a logistic sigmoid function; $\mathbf{x}_t$ is an input vector; $\mathbf{h}_{t-1}$ and $\mathbf{h}_{t-d}$ are recent and delayed controlling states; $d > 1$ is a dilation; $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{U}$ are weight matrices; and $\mathbf{b}$ are bias vectors.

The c -state is calculated from the recent ($\mathbf{c}_{t-1}$), delayed ($\mathbf{c}_{t-d}$) and candidate ($\tilde{\mathbf{c}}_t$) states as follows:

$$\mathbf{c}_t = \mathbf{u}_t \otimes (\mathbf{f}_t \otimes \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \otimes \mathbf{c}_{t-d}) + (1 - \mathbf{u}_t) \otimes \tilde{\mathbf{c}}_t \quad (13)$$

where $\otimes$ denotes the Hadamard product (element-wise product).

Note that the c -state is a weighted combination of past c -states and a new candidate state. Fusion vector $\mathbf{f}_t$ decides in what proportion the recent and delayed c -states are mixed, while update vector $\mathbf{u}_t$ decides about the share of old and new information in the resulting c -state.

Based on state $\mathbf{c}_t$, the output vectors $\mathbf{h}_t$ and $\mathbf{y}_t$ (or $\mathbf{m}_t$) of the cells shown in Fig. 3 are determined as follows:

$$\mathbf{h}_t^{1'} = \mathbf{o}_t^{1'} \otimes \mathbf{c}_t^1, \mathbf{m}_t = [h_{t,1}^{1'}, \dots, h_{t,s_m}^{1'}], \mathbf{h}_t^1 = [h_{t,s_m+1}^{1'}, \dots, h_{t,s_m+s_h}^{1'}] \quad (14)$$

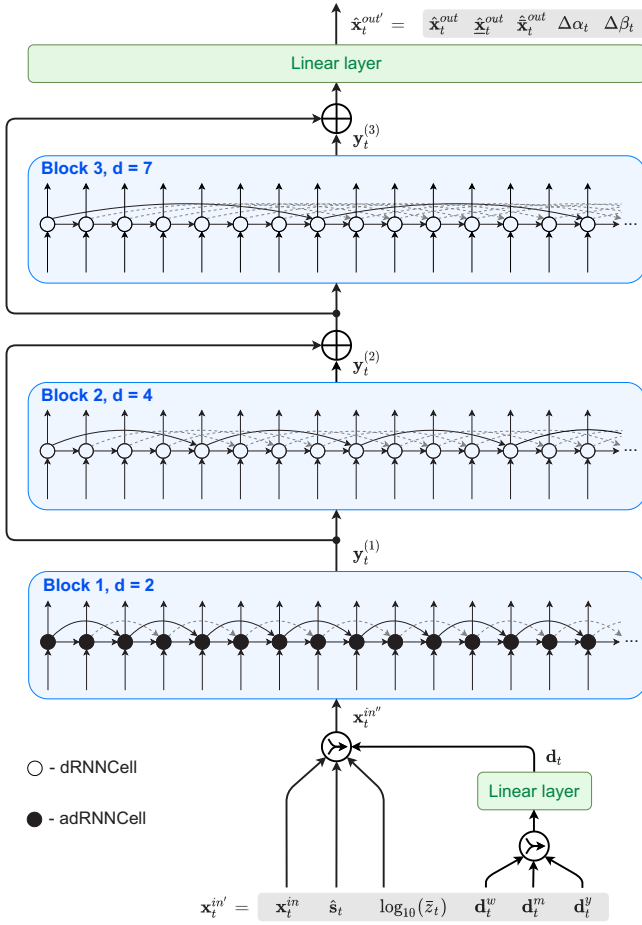


Fig. 4. adRNN architecture.

$$\mathbf{h}_t^{2'} = \mathbf{o}_t^2 \otimes \mathbf{c}_t^2, \mathbf{y}_t = [h_{t,1}^{2'}, \dots, h_{t,s_y}^{2'}], \mathbf{h}_t^2 = [h_{t,s_y+1}^{2'}, \dots, h_{t,s_y+s_q}^{2'}] \quad (15)$$

where s_y, s_h, s_m and s_q denote the lengths of vectors $\mathbf{y}_t, \mathbf{h}_t^1, \mathbf{m}_t$ and $\mathbf{h}_t^2$, respectively (in our implementation s_h and s_q are the same).

Fig. 4 shows the adRNN architecture. It is composed of three single-layer blocks. Block 1 consists of adRNNCells dilated 2, while blocks 2 and 3 consist of dRNNCells dilated 4 and 7, respectively. Note that the cells are fed by delayed states of different dilations in different layers. Delayed connections enable the direct input to the cell of information from a few time steps ago. This can be useful in modeling seasonal dependencies. In our architecture, we use multiple dilated recurrent layers stacked with hierarchical dilations to model the temporal dependencies of different scales. To facilitate the learning process, we use ResNet-style shortcuts between blocks [18].

The inputs representing 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 $\mathbf{d}_t$. This reduces input dimensionality and meaningfully represents sparse binary vectors in the embedding space. The embedding is learned along with the model itself.

The output layer projects linearly vector $\mathbf{y}_t^{(3)}$ produced by the third block to output vector $\hat{\mathbf{x}}_t^{out'}$ composed as follows:

$$\hat{\mathbf{x}}_t^{out'} = [\hat{\mathbf{x}}_t^{out}, \hat{\underline{\mathbf{x}}}_t^{out}, \hat{\bar{\mathbf{x}}}_t^{out}, \Delta\alpha_t, \Delta\beta_t] \quad (16)$$

where $\hat{\mathbf{x}}_t^{out} = [\hat{x}_\tau]_{\tau \in \Delta_t^{out}}$ is a forecasted output pattern; $\hat{\underline{\mathbf{x}}}_t^{out} = [\hat{\underline{x}}_\tau]_{\tau \in \Delta_t^{out}}$ is a vector of lower bounds of PI; $\hat{\bar{\mathbf{x}}}_t^{out} = [\hat{\bar{x}}_\tau]_{\tau \in \Delta_t^{out}}$ is a vector of upper bounds of PI; $\Delta\alpha_t$ and $\Delta\beta_t$ are corrections for smoothing coefficients.

IV. EXPERIMENTAL STUDY

In this section, to evaluate ES-adRNN forecast accuracy, we consider STL for 35 European countries. The data set includes real-world hourly electrical load time series from the period 2016-2018. The data was collected from ENTSO-E repository, www.entsoe.eu/data/power-stats/ (we share this data with the ES-adRNN code in our github repository [16]). The data provides a variety of time series with different properties such as different levels, trends, variance and daily shapes (see Section II in [13] where these time series are analysed). This great variety of time series makes the results more reliable.

We consider a one day-ahead forecasting problem: prediction of the load profile (24 hourly values) for each day of 2018 based on historical data. This was performed for each country with the exception of three countries for which data for the last month (Estonia and Italy) or the last two months (Latvia) of 2018 was unavailable. For these three countries, the test periods were shorter. The ES-adRNN model was optimized on data from the period 2016-17. We perform STL in two variants: using ES-adRNN as an individual model and using an ensemble of five ES-adRNN base models. We denote the latter variant by ES-adRNN_e. 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).

A. Training and Optimization Setup

Our learning process, schedule of hyper-parameter changes is organized around a notion of an epoch. This is usually defined as using all the training data once. Our definition here is based on the number of updates or processed batches, as during training we step l_o times on a batch (with random assignment of series) and for each batch execute a single update based on the average error. Our aim is to define the epoch as the number of updates which brings in a meaningful change in the learning process, (we use 2500), and because the data set contains a small number of series, a single epoch is actually composed with n_o number "sub-epochs", defined in the traditional fashion as one scan of all available data. An additional factor is the batch size: when it grows, the number of updates per sub-epoch diminishes, so the number of the sub-epochs needs to grow. However, in our experience the linear growth is too fast, and we use a sub-linear formula.

During each epoch a number of updates is executed, guided by the average error accumulated by executing l_o (e.g. 50) forward steps, moving by one day, on a batch. The starting point is chosen randomly; the batches include random b series. The model is trained using Adam optimizer.

The d -dilated recurrent cells operate as described above only after d steps, because only after d steps are the delayed states available. Additionally, the Holt-Winters formulas require at least twice the seasonality steps to stabilize, so the system uses several weeks (w_o) at the beginning of each batch as a warm-up period, during which all the ES and RNN calculations take place, with the exception of the training errors, which are not calculated. Similarly, an even longer warm-up period w_s is applied when producing the test results.

We use a schedule of increasing batch sizes and decreasing learning rates proposed in [19]. We start with a small batch size of 2, and increase it, although only once, due to the small number of series, to 5 at epoch 4. We use another schedule of decreasing learning rates, which has a similar, but not exactly the same, effect as increasing the batch size: it allows the validation error to be further reduced. We use the following schedule: $3 \cdot 10^{-3}$ (epochs 1-4), 10^{-3} (epoch 5), $3 \cdot 10^{-4}$ (epoch 6), 10^{-4} (epochs 7-9).

The sizes of c -state and h -state were $s_c = 100$, $s_h = s_q = 40$. These values were obtained by experimentation starting with $s_c = 50$, $s_h = 20$ and doubling it 3 times.

Finally, as described in Section II C, the pinball loss function was utilized, with three different quantile values q , to achieve quantile regression for 0.5, 0.05, and 0.95. The actual values for q^* , q , and $\bar{q}$ were slightly different: 0.485, 0.035, 0.96. These values were arrived at by experimentation, reducing the bias of the center value, and fine-tuning the percentage of exceedance for PIs.

Other hyperparameters were selected as described in [13].

B. Baseline Models

As baseline models we employ:

- Naive – naive model in the form: the forecasted demand profile for day i is the same as the profile for day $i - 7$
- ARIMA – autoregressive integrated moving average model [20],
- ES – exponential smoothing model [20],
- Prophet – modular additive regression model with non-linear trend and seasonal components [21],
- FNM – fuzzy neighborhood model [20]
- GRNN – general regression NN [2],
- MLP – perceptron with a single hidden layer and sigmoid nonlinearities [2],
- SVM – linear epsilon insensitive support vector machine (ϵ -SVM) [22],
- LSTM – long short-term memory [23],
- ANFIS – adaptive neuro-fuzzy inference system [24],
- MTGNN – graph NN for multivariate time series forecasting [25],
- ES-dRNN – hybrid exponential smoothing and dilated recurrent NN model [13],

TABLE I
FORECASTING QUALITY METRICS.

	MAPE	MdAPE	IqrAPE	RMSE	MPE	StdPE
Naive	5.08	4.84	3.32	704.34	-0.26	7.91
ARIMA	3.30	3.01	3.00	475.09	-0.01	5.31
ES	3.11	2.88	2.73	439.26	0.01	5.13
Prophet	4.53	4.32	3.03	619.39	-0.13	6.82
FNM	2.50	2.30	2.29	334.08	-0.11	4.27
GRNN	2.48	2.28	2.27	332.91	-0.11	4.25
MLP	3.05	2.78	2.94	419.01	-0.04	5.07
SVM	2.55	2.29	2.52	357.24	-0.13	4.37
LSTM	2.76	2.57	2.52	381.76	0.02	4.47
ANFIS	3.65	3.17	3.66	507.08	-0.10	6.43
MTGNN	2.99	2.74	2.69	405.18	-0.47	4.85
ES-dRNN	2.33	2.13	2.23	319.04	-0.20	3.90
ES-dRNN _e	2.25	2.05	2.17	309.88	-0.20	3.79
ES-adRNN	2.28	2.08	2.19	315.44	-0.16	3.82
ES-adRNN _e	2.20	2.01	2.13	303.70	-0.13	3.71

- ES-dRNN_e – ensemble of five ES-dRNN base models [13].

The baseline models include classical statistical models (ARIMA, ES), new statistical models (Prophet), nonparametric pattern-based machine learning models (FNM, N-WE), classical machine learning models (GRNN, MLP, SVM, ANFIS) and new recurrent and deep NN architectures (LSTM, MTGNN). They also include the predecessor of the proposed model, i.e. ES-dRNN and its ensemble variant ES-dRNN_e.

C. Results

Table I summarizes the forecasting quality metrics averaged over the 35 countries. It is clear from this table that ES-adRNN_e outperforms all other models in terms of accuracy. It shows the lowest MAPE, MdAPE and RMSE. To confirm the best performance of ES-adRNN_e, we perform a pairwise one-sided Giacomini-White test (GM test) for conditional predictive ability [26] (we used the multivariate variant of the GW test implemented in <https://github.com/jeslago/epftoolbox> [27]). Fig. 5 demonstrates results of this test as a heat map representing the obtained p -values. 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. 5 clearly show the state-of-the-art performance of ES-adRNN_e and ES-adRNN. ES-adRNN_e performed significantly better in terms of accuracy than all the other comparative models. ES-adRNN is second only to the ES-dRNN_e.

IqrAPE and StdPE shown in Table I measure the dispersion of APE and PE, respectively. Note that ES-adRNN_e produced the least dispersed forecasts. The individual version, ES-adRNN, is close behind it. MPE is a measure of the forecast bias. Note that most of the models including ES-adRNN and ES-adRNN_e produced negatively biased forecasts, which means overprediction. The proposed model is equipped with a mechanism for controlling the bias included in the loss function (7), so the bias can be reduced. But reduction in bias

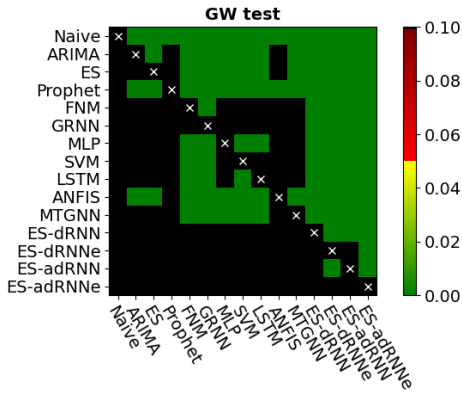


Fig. 5. Results of the Giacmini-White test.

can lead to a decrease in forecast accuracy due to overfitting, so we purposely avoided it.

Fig. 6 shows several examples of forecasted daily patterns. PIs are also shown for ES-adRNNe. To evaluate the accuracy of the PIs predicted by this model, we calculate the number of forecasts lying inside, above and below of their PIs. We achieved: $90.18\% \pm 2.86\%$ forecasts are inside PIs, $4.74\% \pm 1.47\%$ are below PIs and $5.08\% \pm 1.61\%$ are above PIs. These values correspond to the assumed PIs of 90% with lower and upper bounds $\underline{q} = 0.05$ and $\bar{q} = 0.95$, respectively.

V. CONCLUSION

STLF is challenging due to multiple seasonality, nonlinear trend and variable variance. To deal with this problem our model combines ES for time series preprocessing and RNN for capturing both short and long-term dependencies in time series. It is equipped with many useful mechanisms and procedures such as cross-learning on many time series, common learning procedure for ES and RNN, adjusting of ES parameters by RNN, on-the-fly deseasonalization, dilated recurrent cells, ResNet-style shortcuts, and extended input vectors.

In this work, we extend further the model by adding dynamic attention. We introduce a new type of the gated recurrent cell, adRNNCell, which implements an attention mechanism for weighting the input information. This mechanism permits the cell to utilize the most relevant components of the input patterns in a flexible manner to improve the forecasting performance of the model.

As the experimental study showed, dynamic attention significantly improves the accuracy of forecasting. The proposed ES-adRNNe outperformed statistical and machine learning models including its predecessor ES-dRNNe. Note that ensembling in our approach does not require additional effort related to the selection of additional hyperparameters such as a parameter for controlling learners' diversity. The diversity is provided by the random initialization of the models. The great advantages of ES-adRNNe are its ability to deal with raw time series, without requiring their decomposition, and its ability to produce both point forecasts and PIs. PIs with a

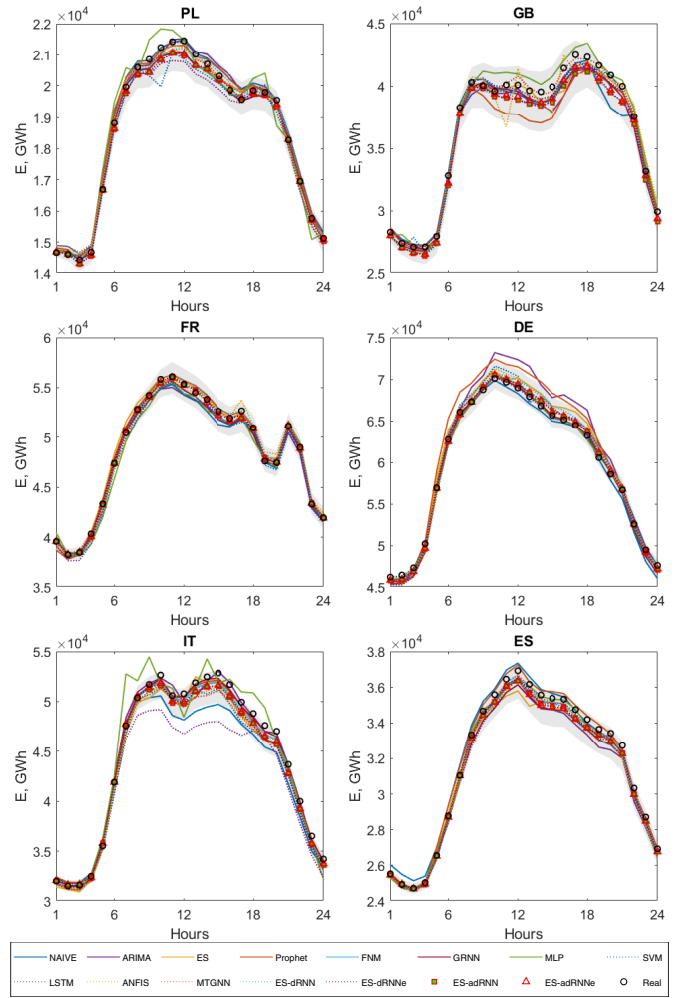


Fig. 6. Examples of the forecasted daily profiles. 90% PIs for ES-adRNNe are shown as gray-shaded areas.

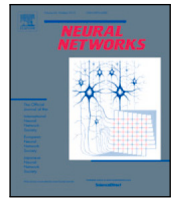
specified probability coverage give very valuable information about the uncertainty of the prediction.

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] K. Benidis, S.S. Rangapuram, V. Flunkert, B. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, L. Callot, T. Januschowski, "Neural forecasting: Introduction and literature overview," *arXiv:2004.10240*, 2020.
- [2] G. Dudek, "Neural Networks for Pattern-based short-term load forecasting: A comparative study," *Neurocomputing*, vol. 2015, pp. 64-74, 2016.
- [3] G. Dudek, "Randomized neural networks for forecasting time series with multiple seasonality," in *proc. 16th International Work-Conference on Artificial Neural Networks, IWANN 2021*, Springer LNCS 12862, pp. 196-207, 2021.
- [4] Z.A. Bashir and M.E. El-Hawary, "Applying wavelets to short-term load forecasting using PSO-based neural networks," *IEEE Transactions on Power Systems*, vol. 24, no. 1, pp. 20-27, 2009.

- [5] H.S. Hippert, J.W. Taylor, "An evaluation of Bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting," *Neural Networks*, vol. 23, pp. 386-395, 2010.
- [6] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu and J. He, "Short-term load forecasting with deep residual networks," *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3943-3952, 2019.
- [7] H.J. Sadaei, P.C. de Lima e Silva, F.G. Guimarães, M.H. Lee, "Short-term load forecasting by using a combined method of convolutional neural networks and fuzzy time series," *Energy*, vol. 175, pp. 365-377, 2019.
- [8] X. Kong, C. Li, F. Zheng and C. Wang, "Improved deep belief network for short-term load forecasting considering demand-side management," *IEEE Transactions on Power Systems*, vol. 35, no. 2, pp. 1531-1538, 2020.
- [9] H. Hewamalage, C. Bergmeir and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, vol. 37, no. 1 pp. 388-427, 2021.
- [10] S. Wang, X. Wang, S. Wang and D. Wang, "Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting," *International Journal of Electrical Power & Energy Systems*, vol. 109, pp. 470-479, 2019.
- [11] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841-851, 2019.
- [12] J. Li et al., "A Novel Hybrid Short-Term Load Forecasting Method of Smart Grid Using MLR and LSTM Neural Network," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2443-2452, 2021.
- [13] S. Smyl, G. Dudek, P. Pelka, "ES-dRNN: A Hybrid Exponential Smoothing and Dilated Recurrent Neural Network Model for Short-Term Load Forecasting," *arXiv:2112.02663*, 2021.
- [14] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, pp. 75-85, 2020.
- [15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems, NIPS*, 2017.
- [16] ES-adRNN code and data: <https://github.com/slaweck17/ES-adRNN>.
- [17] G. Dudek, P. Pelka and S. Smyl, "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.
- [18] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in proc. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770-778, 2016.
- [19] S.L. Smith, P.J. Kindermans, C. Ying, Q.V. Le, "Don't decay the learning rate, increase the batch size," in *Proc. ICLR*, 2018. Available at <https://openreview.net/pdf?id=B1Yy1BxCZ>.
- [20] G. Dudek, "Pattern similarity-based methods for short-term load forecasting – part 2: Models," *Applied Soft Computing*, vol. 36, pp. 422-441, 2015.
- [21] S.J. Taylor, B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37-45, 2018.
- [22] P. Pelka, "Pattern-based forecasting of monthly electricity demand using support vector machine," in proc. *International Joint Conference on Neural Networks, IJCNN 2021*, pp. 1-8, 2021.
- [23] P. Pelka, G. Dudek, "Pattern-based long short-term memory for mid-term electrical load forecasting," in proc. *International Joint Conference on Neural Networks, IJCNN 2020*, pp. 1-8, 2020.
- [24] P. Pelka, G. Dudek, "Neuro-fuzzy system for medium-term electric energy demand forecasting," in proc. *38th International Conference on Information Systems Architecture and Technology, ISAT 2018*, Springer AISC 655, pp. 38-47, 2018.
- [25] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in proc. *26th International Conference on Knowledge Discovery & Data Mining, ACM SIGKDD*, 2020.
- [26] R. Giacomini, H. White, "Tests of conditional predictive ability," *Econometrica*, vol. 74(6), pp. 1545-1578, 2006.
- [27] J. Lago, G. Marcjasz, B. De Schutter, R. Weron, "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark," *Applied Energy*, vol. 293, pp. 116983, 2021.



Full Length Article

Contextually enhanced ES-dRNN with dynamic attention for short-term load forecasting

Slawek Smyl^a, Grzegorz Dudek^{b,*}, Paweł Pełka^b^a Meta, 1 Hacker Way, Menlo Park, CA 94025, USA^b Department of Electrical Engineering, Czestochowa University of Technology, 42-200 Czestochowa, Al. Armii Krajowej 17, Poland

ARTICLE INFO

Keywords:

Exponential smoothing
Hybrid forecasting models
Recurrent neural networks
Short-term load forecasting
Time series forecasting

ABSTRACT

In this paper, we propose a new short-term load forecasting (STLF) model based on contextually enhanced hybrid and hierarchical architecture combining exponential smoothing (ES) and a recurrent neural network (RNN). The model is composed of two simultaneously trained tracks: the context track and the main track. The context track introduces additional information to the main track. It is extracted from representative series and dynamically modulated to adjust to the individual series forecasted by the main track. The RNN architecture consists of multiple recurrent layers stacked with hierarchical dilations and equipped with recently proposed attentive dilated recurrent cells. These cells enable the model to capture short-term, long-term and seasonal dependencies across time series as well as to weight dynamically the input information. The model produces both point forecasts and predictive intervals. The experimental part of the work performed on 35 forecasting problems shows that the proposed model outperforms in terms of accuracy its predecessor as well as standard statistical models and state-of-the-art machine learning models.

1. Introduction

Electricity demand forecasting is extremely important for the energy sector to ensure the secure, effective and economic operation of the power system. The need for forecasting results from the fact that electricity cannot be stored on a massive scale, so the supply must be adequate to meet all customers' demand simultaneously, instantaneously, and reliably. To provide electricity on demand, electric system operations have to be planned and conducted with those goals in mind.

Long- and medium-term load forecasting predict the power system load over time ranges measured in months or years, while short-term load forecasting (STLF) concerns a forecast horizon of a few hours to a few days. STLF is necessary to schedule generation resources to meet the future load demand and to optimize the power flow on the transmission network to reduce congestion and avoid overloads. It is used for unit commitment, generation dispatch, hydro scheduling, hydrothermal coordination, spinning reserve allocation, interchange and low flow evaluation, fuel allocation and network diagnosis. Improving the accuracy of forecasting can significantly reduce the operating costs of the power system and increase its security. As electricity pricing is driven by supply and demand, STLF plays a key role in competitive electricity markets. It determines the financial performance of the market participants such as generation and transmission utilities, electricity

traders and financial institutions. Accurate forecasts translate directly into reduced financial risks and improved financial performance.

STLF is a difficult problem due to the complexity of electricity demand time series, which can express nonlinear trend, multiple seasonality (yearly, weekly and daily), variable variance, random fluctuations and changing daily profiles. They are influenced by many stochastic factors such as climatic, weather and economic conditions. The increasing contribution of volatile, fluctuating renewable energy sources, which has been observed in recent years, introduces an additional random component to electricity demand time series. The complexity of the problem and its importance for the safe, reliable and effective operation of power systems and electricity markets have generated great interest among researchers in this field.

1.1. Related work

STLF complexity places high demands on forecasting models. They have to deal with non-stationarity and multiple seasonality in time series. Roughly, STLF methods can be classified into statistical/econometric and more sophisticated machine learning (ML) methods. The first category has numerous disadvantages, such as linear nature, limited adaptability, limited ability to deal with complex seasonal patterns, problems with introducing exogenous variables into the model and

* Corresponding author.

E-mail addresses: slawek.smyl@gmail.com (S. Smyl), grzegorz.dudek@pcz.pl (G. Dudek), pawel.pelka@pcz.pl (P. Pełka).<https://doi.org/10.1016/j.neunet.2023.11.017>

Received 23 December 2022; Received in revised form 5 September 2023; Accepted 7 November 2023

Available online 8 November 2023

0893-6080/© 2023 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC license (<http://creativecommons.org/licenses/by-nc/4.0/>).

capturing long-term relationships in time series (Smyl, Dudek, & Peřka, 2021). Typical representatives of this category are: auto-regressive integrated moving average (ARIMA) (Arora & Taylor, 2018), exponential smoothing (ES) (Taylor, 2012), linear regression (Charlton & Singleton, 2014), and Kalman filtering (Takeda, Tamura, & Sato, 2016).

ML and computational intelligence methods are more flexible in modeling complicated nonlinear relationships than statistical approaches. They include well-known ML methods such as classical neural networks (NNs) (Dudek, 2016), tree-based models (Aprillia, Yang, & Huang, 2020), support vector machine (Li, Shi, Sibtain, Li, & Mbanze, 2020), neuro-fuzzy systems (Kandilogiannakis, Mastorocostas, & Voulodimos, 2022) as well as the latest developments in the field of deep learning (Hong, Zhou, Li, Xu, & Zheng, 2020), recurrent NNs (Kong et al., 2017), boosted decision trees (Wang et al., 2020), hybrid and ensemble models (Massaoudi et al., 2021; Tan et al., 2019).

In recent years, the forecasting literature has mainly focused on NNs, especially deep, recurrent and hybrid solutions, but also tree-based models should be mentioned due to their high performance in forecasting competitions (Bojer & Meldgaard, 2021). The success of deep NNs (DNNs) can be largely attributed to their increased complexity which allows the model to learn the appropriate data representation for modeling the target function. While the success of RNNs is due to their ability to capture complex, short- and long-term temporal relationships in time series and thus model the dynamics of the series more accurately.

New ideas and achievements in deep learning can be immediately assimilated by STLF models. Some examples of STLF models employing state-of-the-art RNN and DNN are: Eskandari, Imani, and Moghaddam (2021), where convolutional NNs (CNNs) were used to extract load and temperature features that feed the bidirectional propagating RNN to make hourly electrical load forecasts; Shi, Xu, and Li (2017), where pooling-based deep learning is proposed, which batches a group of customers' load profiles into a pool of inputs and is able to learn the uncertainty of load profiles; Kim, Moon, Hwang, and Kang (2019), where a recurrent inception CNN is proposed for STLF that combines RNN and 1-dimensional CNN; Wang, Wang, Wang, and Wang (2019), where a bi-directional long short-term memory NNs with attention mechanism is proposed; Chen et al. (2018), where deep residual networks integrate domain knowledge and researchers' understanding of the problem and produce probabilistic load forecasts using Monte Carlo dropout; and Lin and Wu (2021), where an attentive transfer learning based graph NN is proposed that can utilize the learned prior knowledge to improve the learning process for residential electric load forecasting.

Many of the above-mentioned STLF models use ensembling or hybridization to increase forecast accuracy and robustness. Combining base models in the ensemble can take many forms. For example in Dudek, Peřka, and Smyl (2022), to reduce the model variance related to the stochastic optimization algorithm as well as data and parameter uncertainty, three levels of ensembling were used at the same time: stage of training level, data subset level, and model level. In Yang, Guo, Sun, Han, and Wang (2022), an ensemble of RNNs is applied, which learns on the components of the bivariate empirical mode decomposition, to forecast the interval-valued load. Another example of an ensemble based on the products of time series decomposition can be found in Gao, Du, Suganthan, Zhou, and Yuen (2022), where an ensemble of randomized DNN combined with an empirical wavelet transformation is proposed. In Piotrowski, Baczyński, Kopyt, and Gulczyński (2022), several ensemble methods aggregating different ML models are compared for one-day-ahead wind power forecasting.

Hybrid models that combine two or more base models into one take advantages of the best features of each base model while eliminating their drawbacks. The beneficial effects of the model hybridization for STLF were demonstrated in many papers. For example, in the approach proposed in Smyl et al. (2021), a statistical method (ES) helps in data preprocessing for ML model (RNN). Namely, ES extracts dynamically the main components of each individual time series and

enables appropriate series representation for RNN. This makes RNN capable of dealing with complex series expressing multiple seasonality. In Wang et al. (2020), a temporal CNN is utilized to extract hidden information and long-term temporal relationships in the data and a boosted tree model (LightGBM) is applied to predict future loads based on the extracted features. The approach presented in Massaoudi et al. (2021) combines three models: LightBoost, XGBoost and NN. The first two models generate the metadata while NN uses them to calculate the final predictions.

1.2. Motivation, inspiration and contribution

Existing ML models have several drawbacks and limitations we try to improve in this study. They include:

- Classical ML models, non-recurrent ones, have difficulty in modeling temporal relationships in sequential data. The training samples are unrelated and presented in random order during training. Our model uses RNNs, which can model temporal relationships due to their internal states.
- ML models usually require data preprocessing to simplify the forecasting problem, such as time series decomposition, deseasonalization, or stationarization. Our model is able to deal with the raw time series, without requiring their decomposition or any preprocessing. The preprocessing component preprocesses time series dynamically on-the-fly using data extracted from the series and produced by the ES component.
- Typical RNNs have difficulty in modeling long-term and seasonal relationships. To overcome this issue, we use dilated recurrent cells, which are fed by both recent and delayed states. Our hierarchical RNN architecture extends their receptive fields in subsequent layers to capture short-term, long-term, and seasonal dynamics.
- Traditional ML models require additional procedures to select input information. However, our recurrent cells have a built-in mechanism to dynamically weigh input information (internal attention).
- Typically ML models produce point forecasts. Our model is able to produce both point forecasts and predictive intervals of assumed width.
- Most ML models learn on a single time series. Even if they learn on all series (global models) they “see” each series one at a time, while in our model every time series is accompanied by the context.
- Most ML models forecast a single values, e.g. the expected value, so they are of single-step ahead type. In contrast, our model is designed to have a vector output, allowing it to generate forecasts for multiple time steps ahead simultaneously. So, it is able to forecast 24 steps ahead at once.
- Very few ML models have a component dealing with seasonality. Our model deals with seasonality using ES. It produces a seasonal component, which is used for deseasonalization and also feeds into RNN.
- Existing forecasting models often lack mechanisms to effectively regulate forecast bias. However, our model addresses this limitation by incorporating a mechanism to mitigate forecast bias through the use of a quantile loss function.

Moreover, global ML forecasting models learn from all series in a dataset, however when forecasting a particular series, only inputs derived from this one series are typically used. For every series there is likely to exist one or more series in the dataset, that, if used as an additional input, would improve the forecasting accuracy. These issues have been researched in depth. In the context of statistical models we have the notion of Granger causality (Granger, 1969), there are factor models (De Stefani & Bontempi, 2021; Satorras, Rangapuram, &

Januschowski, 2022; Sen, Yu, & Dhillon, 2019; Wang et al., 2019) and in ML there has been quite a bit of work done on Graphical NNs (Li, Wang, Zhu, & Huang, 2018; Tran, Mathews, Ong, & Xie, 2021; Wu et al., 2020a; Zhou et al., 2020). In the ML context, there are two main issues here: how to find a small subset of “friendly” time series out of many, perhaps dozen or hundreds of thousands of others, and having found them, how to aggregate the features extracted, considering that just concatenating them is unlikely to succeed, because in ML systems inputs are position sensitive. Additionally, if, as is typical, the number of “friendly” series, or neighborhood size, is a constant, it is a rigid constraint, too low for some series and too high for other series.

We propose an RNN-based forecasting system composed of two tracks: the first one, the context track, processes a representative subset of series and outputs an opaque vector that, optionally after being modulated to suit a particular series, is concatenated with other features and becomes an input to the second, main forecasting track. The idea of context has been inspired by Qin et al. (2017), but the context here is calculated in a quite different, and much more computationally efficient way, notably, it does not use standard attention (distinct from a completely different attention mechanism used internally by adRNNCell, see below).

The architecture of each track is inspired by the winning submission to the M4 forecasting competition 2018 (Smyl, 2020). This hybrid and hierarchical forecasting model was composed of ES and RNN modules that are simultaneously trained by the same optimization algorithm. ES allows for on-the-fly preprocessing of the series, i.e. deseasonalization and adaptive normalization, and thus captures the main features of the individual series. RNN is composed of several stacked long short-term memory cells (LSTMs) with dilated and residual connections, which improve long-term memory performance and speed-up training. Due to the hybrid and hierarchical architecture and cross-learning on many series, the model is able to learn both local and global time series features. This and also ensembling at three levels improve forecasting accuracy.

Based on the winning submission to the M4 competition, in Dudek et al. (2022), we developed a model for mid-term load forecasting and in Smyl et al. (2021) we developed a model for STLTF. The latter model uses a new type of recurrent cell with additional delayed states and an additional output vector. The delayed states help to model seasonalities and long-term relationships while introducing an additional output vector allows the cell to better control the flow of information. An interesting feature of the model is that RNN learns not only the forecasting function but also the ES parameters to ensure optimal time series preprocessing. A further refinement of the STLTF model proposed in Smyl, Dudek, and Pelka (2022) is the introduction of an internal attention mechanism that dynamically weights the inputs for the cell.

In continuing the development of the STLTF models based on a hybrid and hierarchical architecture combining ES and RNN, this study makes the following contributions:

- (1) We propose a new architecture of the hybrid and hierarchical model combining ES and RNN, which consists of two tracks: the context track and the main track. The context track generates a dynamic context vector based on the historical data of a representative group of series. This vector is adjusted to the forecast series and used as an additional input to the main track RNN. It extends per-series input data with the information contained in the context series and thus supports forecasting individual series.
- (2) To validate the efficacy of our proposed approach, we conduct extensive STLTF experiments for 35 European countries. The experimental results show the high performance of the proposed model, which outperforms in terms of accuracy its predecessor and 16 baseline models, including both statistical and ML state-of-the-art models.

Table 1

Abbreviations.

Abbreviations	Definitions
adRNNcell	Attentive dilated recurrent neural network cell
ARIMA	Auto-regressive integrated moving average
cES-adRNN	Contextually enhanced hybrid model combining exponential smoothing and dilated recurrent neural network with attention mechanism
CNN	Convolutional neural network
DNN	Deep neural network
dRNNcell	Dilated recurrent neural network cell
ES	Exponential smoothing
ES-adRNN	Hybrid model combining exponential smoothing and dilated recurrent neural network with attention mechanism
ES-dRNN	Exponential smoothing dilated recurrent neural network
IqrAPE	Interquartile range of absolute percentage error
LSTM	Long short-term memory
MAPE	Mean absolute percentage error
MdAPE	Median of absolute percentage error
ML	Machine learning
MPE	Mean percentage error
NN	Neural network
PE	Percentage error
PI	Predictive interval
RMSE	Root mean square error
RNN	Recurrent neural network
StdPE	Standard deviation of percentage error
STLF	Short-term load forecasting

The proposed model inherits the features of its predecessors, including attentive dilated recurrent cells, multiple dilated recurrent layers stacked with hierarchical dilations, cross-learning ability, ability to learn ES smoothing coefficients, and ability to produce both point forecasts and predictive intervals (PIs). The proposed architecture is flexible and can be applied to other time series forecasting tasks beyond the STLTF. We provide the open-source code of our model in the GitHub repository ([cES-adRNN code and data, 2022](#)).

The rest of the paper is structured as follows. First, the forecasting model is described in Section 2, which includes the STLTF problem, model architecture and components, and loss function. Then, an extensive experimental evaluation is provided in Section 3. Finally, conclusions are drawn in Section 4. Table 1 lists key abbreviations.

2. Forecasting model

In this section we define the forecasting problem and present the proposed solution. We describe the functions of the context and main tracks, the model architecture and its components, and the loss function which enables the model to produce both point forecasts and predictive intervals (PIs).

2.1. STLTF problem statement

The goal of STLTF is to use the previously observed loads of the power system to forecast a fixed length of the future loads. In this study, we consider hourly load time series and a 24-hour forecast horizon. Thus the goal is to predict $\{z_\tau\}_{\tau=M+1}^{M+24}$, given a sequence of past observations, $\{z_\tau\}_{\tau=1}^M$, where z_τ is the system load at hour τ and M is the time series length (training part).

This STLTF problem is challenging because the load time series expresses a nonlinear trend, varying variance, random fluctuations and triple seasonality: daily, weekly and yearly. The daily load profiles differ in shape depending on the day of the week and the season of the year. The load time series nature is specific for different countries. It is shaped by socioeconomic factors such as population, urbanization, industrialization, level of technological development, electricity price as well as climatic and weather conditions. To get a broader picture of the load time series characteristics for European countries see Smyl et al. (2021), where we analyzed load series in terms of variability for various horizons, seasonal components and similarity of daily profiles.

2.2. Context track and main track

The proposed forecasting model is composed of two tracks: the context track and the main forecasting track, see Fig. 1. A helpful context to a model forecasting a series should be one that extracts some kind of overview of the history of a representative subset, perhaps all, of a series, up to now. However, if the number of time series is large, say hundreds or more, it is not feasible to concatenate them all to form a single input. We reduce the severity of this problem by using a context model that processes just one single series at a time, however within the batch structure, for purposes of efficiency, context series are processed in parallel. At the end of each step through time, we flatten the batch output into one vector, i.e. context vector r .

Forecasting follows the usual stages of on-the-fly preprocessing (see Smyl et al., 2021, 2022), which includes adaptive normalization and deseasonalization, but then, before reaching the RNN, the input is augmented (concatenated) with the output of the context track, which can be the same for each series in a batch being forecasted. However, if the number of series is relatively small, which means each series can be presented to the forecasting network many, say more than 20, times during training, we can improve the accuracy by adding per-series parameters, i.e. vector g of the same length as of the context vector r , initialized as ones. Vector g will modulate the general context information in a way to match particular series needs. This vector is not dynamic, i.e. does not change while stepping through the batch time steps. If the number of series were larger, which means it would be difficult to fit per-series parameters, we could try to calculate vector g dynamically, using some kind of attention mechanism, but in our preliminary exploration this resulted in worse forecasting accuracy, and in this work, we do not report further on this possibility.

Both context and forecasting tracks are synchronized in their steps through time. All the parameters, of the context track and main track, and also per-series parameters, if used, are fitted in order to lower the forecasting error. Note that there is no separate loss function applied to the output of the context track.

A batching mechanism allows us to use a relatively large number of series as context series, but of course, there are limitations on the size of one batch. One way to reduce the number of context series is to use business knowledge and choose a set representing well all the series (and to make things easier, without missing data). Another way would be to create a factor model, where series are modeled as a linear combination of some limited number of base series, and then use the bases as context series.

2.3. Model architecture

Both tracks, context and main, are based on ES-adRNN, i.e. a hybrid model combining ES with RNN, which is equipped with attentive dilated cells (adRNNcells) (Smyl et al., 2022). The ES-adRNN architecture includes four components: ES component, preprocessing component, RNN component and postprocessing component, which is only used in the main track.

The time series denoted in Fig. 1 as Z_c (context time series) and Z (forecasted time series) are decomposed individually by ES into seasonal and level components. The seasonal components, S , are used by the preprocessing component to deseasonalize time series. Generally, run-time preprocessing prepares training sets for RNN in the context track and main track, Ψ_c and Ψ , respectively. The training sets include preprocessed time series and calendar variables D . Additionally, training set Ψ includes modulated context vectors R' .

RNN in the main track learns the function mapping input vectors into output patterns which are composed of three parts: (i) vector of encoded point forecasts of hourly loads for the next day, (ii) two vectors encoding predictive intervals (PIs), and (iii) the corrections of ES smoothing parameters, $\Delta\alpha$ and $\Delta\beta$. RNN in the context track, based on input vectors of similar structure to those in the main track

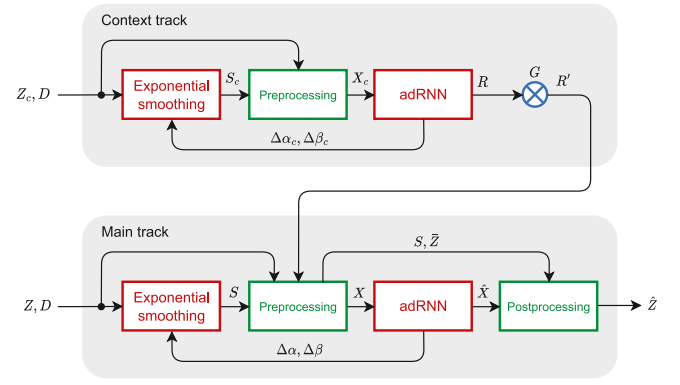


Fig. 1. Block diagram of the proposed forecasting model.

except the context part, learns the context vectors R . These vectors are modulated by vectors G to adjust them to the individual time series in the main batch. A postprocessing component in the main track transforms encoded vectors of point forecasts and PIs, $\hat{X}$, into real values, $\hat{Z}$. To do so, it uses seasonal components S and average values of input sequences $\bar{Z}$ determined by the ES and preprocessing components.

2.4. Exponential smoothing component

ES component is a simplified Holt-Winters model with multiplicative seasonality. It is composed of two equations expressing a level and weekly seasonality (Smyl et al., 2022):

$$\begin{aligned} l_{t,\tau} &= \alpha_t \frac{z_\tau}{s_{t,\tau}} + (1 - \alpha_t) l_{t,\tau-1} \\ s_{t,\tau+168} &= \beta_t \frac{z_\tau}{l_{t,\tau}} + (1 - \beta_t) s_{t,\tau} \end{aligned} \quad (1)$$

where $l_{t,\tau}$ is a level component, $s_{t,\tau}$ is a weekly seasonal component, and $\alpha_t, \beta_t \in [0, 1]$ are smoothing coefficients, all for recursive step t .

The unique feature of model (1) is that the smoothing coefficients are not constant as in the standard solution, but they change at each step t , adapting dynamically to the current time series properties. The smoothing coefficients are adapted using corrections $\Delta\alpha_t$ and $\Delta\beta_t$ learned by RNN as follows (Smyl et al., 2022):

$$\begin{aligned} \alpha_{t+1} &= \sigma(I\alpha + \Delta\alpha_t) \\ \beta_{t+1} &= \sigma(I\beta + \Delta\beta_t) \end{aligned} \quad (2)$$

where $I\alpha$ and $I\beta$ are initial values of the smoothing coefficients, and σ is a sigmoid function, which maintains the coefficients within a range from 0 to 1.

2.5. Preprocessing and postprocessing components

The goal of preprocessing is twofold. First, it converts the time series to a form that is more convenient for processing by RNN. Second, it creates the input and output patterns and composes them into training sets.

The primary input for both the context track and main track is the weekly sequence of series preceding directly the forecasted daily period. Let Δ_t^{in} of size 168 h be the window covering the t th input weekly sequence, and Δ_t^{out} of size 24 h be the window covering the following daily sequence, see Fig. 2. Both windows are shifted by 24 h to generate subsequent input and output sequences. The input sequence is deseasonalized, normalized and squashed as follows:

$$x_\tau^{in} = \log \frac{z_\tau}{\bar{z}_t \hat{s}_{t,\tau}} \quad (3)$$

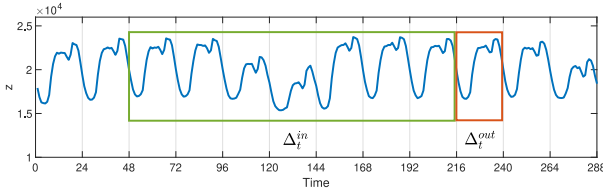


Fig. 2. Sliding windows defining input and output sequences.

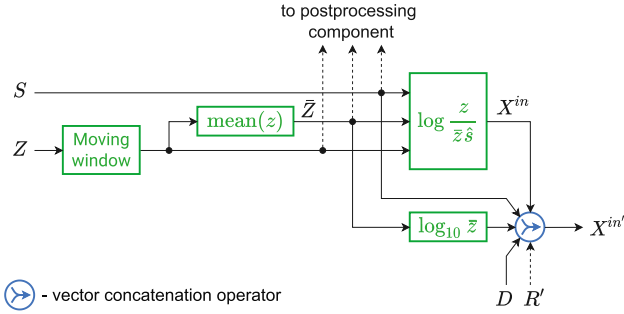


Fig. 3. Preprocessing of the input data and generating the input pattern (the links shown with dashed lines do not exist in the preprocessing component for the context track).

where $\tau \in \Delta_t^{in}$, $\bar{z}_t$ is the average value of the input sequence, and $\hat{s}_{t,\tau}$ is the seasonal component predicted by ES for τ in step t .

The preprocessed input sequence is represented by vector $\mathbf{x}_t^{in} = [\mathbf{x}_t^{in}]_{\tau \in \Delta_t^{in}} \in \mathbb{R}^{168}$. The purpose of squashing with a log function is to prevent outliers from disrupting the learning process. Deseasonalization using $s_{t,\tau}$ removes weekly seasonality while normalization using $\bar{z}_t$ removes the long-term trend in the input window. Thus all preprocessed series are brought to the same level, which is very important when cross-learning is applied. Note that input patterns $\mathbf{x}_t^{in}$ have a dynamic character. They are adapted in each training epoch due to the adaptation of seasonal component $s_{t,\tau}$. This can be seen as the learning of the most useful representation for RNN.

To enrich the input information, the input patterns for the main track are extended with the level and seasonality of the series, calendar data as well as modulated context vector:

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

where $\hat{s}_t \in \mathbb{R}^{24}$ is a vector of 24 seasonal components predicted by ES for $\tau \in \Delta_t^{out}$ reduced by 1, $\mathbf{d}_t^w \in \{0, 1\}^7$, $\mathbf{d}_t^m \in \{0, 1\}^{31}$ and $\mathbf{d}_t^y \in \{0, 1\}^{52}$ are binary one-hot vectors encoding day of the week, day of the month and year for the forecasted day, respectively, and $\mathbf{r}_t'$ is a modulated context vector from the context track (see Section 2 G).

An enriched input pattern for the context track has a similar structure to (4) except for the modulated context vector. Calendar variables, $\mathbf{d}_t^w$, $\mathbf{d}_t^m$, $\mathbf{d}_t^y$, carry information about the location of the forecasted daily sequence in the weekly and yearly cycles and help to deal with fixed-date public holidays. Component $\log_{10}(\bar{z}_t)$ informs about the local level of the series, while $\hat{s}_t$ expresses the daily variability of the forecasted sequence. The modulated context vector, $\mathbf{r}_t'$, introduces additional information from the context series for step t adjusted to the forecasted time series. Fig. 3 shows the preprocessing of the input data and generating the input pattern.

The output pattern represents the target daily sequence covered by Δ_t^{out} . To obtain the output pattern, $\mathbf{x}_t^{out} = [\mathbf{x}_t^{out}]_{\tau \in \Delta_t^{out}} \in \mathbb{R}^{24}$, the original sequence is normalized as follows:

$$x_\tau^{out} = \frac{z_\tau}{\bar{z}_t} \quad (5)$$

where $\tau \in \Delta_t^{out}$.

Note that to define output patterns, we only use normalization so that the errors determined by the loss function for different time series (of different levels) are comparable and have a similar impact on learning. The output patterns are defined only for time series processed in the main track (the context track produces context vectors $\mathbf{r}$, which have no target values). As targets, they are compared with the patterns predicted by the main track and the resulting error is used by the optimization algorithm to adapt all model parameters including both main and context tracks parameters, i.e. ES and RNN parameters as well as modulation vectors $\mathbf{g}$.

The model is trained on the training patterns produced by shifting by 24 h the adjacent moving windows, Δ_t^{in} and Δ_t^{out} . RNN in the main track predicts a vector corresponding to the 24 point forecasts for the next day: $\hat{\mathbf{x}}_t^{RNN} = [\hat{x}_\tau^{RNN}]_{\tau \in \Delta_t^{out}} \in \mathbb{R}^{24}$. The components of this vector are converted by the postprocessing module into real values using transformed Eq. (3):

$$\hat{z}_\tau = \exp(\hat{x}_\tau^{RNN}) \bar{z}_t \hat{s}_{t,\tau} \quad (6)$$

where $\tau \in \Delta_t^{out}$.

However, the loss function uses its normalized version, matching (5):

$$\hat{x}_\tau^{out} = \frac{\hat{z}_\tau}{\bar{z}_t} \quad (7)$$

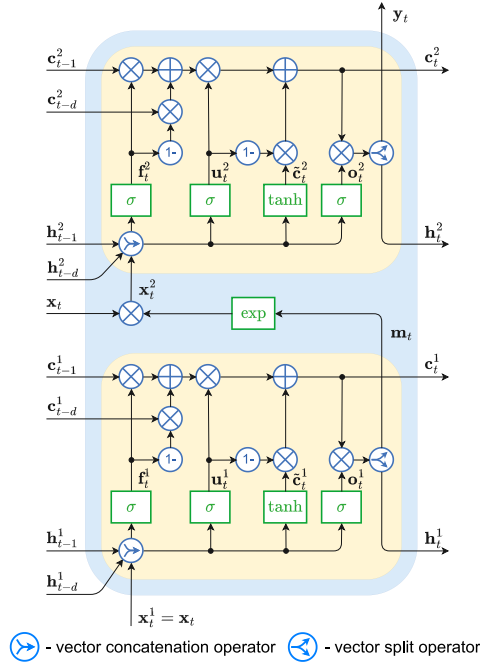
RNN also predicts two vectors representing bounds of PI: $\hat{\mathbf{x}}_t^{RNN}$ and $\hat{\mathbf{x}}_t^{RNN}$. They are converted into real values in the same way as the point forecasts, using (6).

2.6. RNN component

RNN uses our previously proposed LSTM-like cell, adRNNCell, described in Smyl et al. (2022). It was designed to deal with multiple seasonality in time series and is equipped with a built-in attention mechanism for weighing the input information. adRNNCell is composed of two identical dRNNCells as shown in Fig. 4.

The characteristic features of adRNNCell can be summarized as follows:

- Each dRNNCell has two cell states (c -states) and two controlling states (h -states). The recent states, $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$, introduce information to the cell from the recent step $t-1$, as in the standard LSTM solution (Hochreiter & Schmidhuber, 1997), while the dilated states, $\mathbf{c}_{t-d}$ and $\mathbf{h}_{t-d}$, introduce information from the delayed step $t-d$, $d > 1$. These delayed states extend the “receptive field” of the cell which facilitates modeling long-term and seasonal dependencies.
- dRNNCell is equipped with two weighting mechanisms, derived from GRU (Cho et al., 2014), which manage the c -state. The first one, controlled by the f -gate, weights recent and delayed c -states, $\mathbf{c}_{t-1}$ and $\mathbf{c}_{t-d}$, while the second one, controlled by the u -gate, weights old and new c -states, i.e. combination of $\mathbf{c}_{t-1}$ with $\mathbf{c}_{t-d}$ and $\bar{\mathbf{c}}_t$. Effectively using both recent and delayed states allows the cell to have a longer memory, and when used as part of a multi-cell RNN with varying dilations, gives the RNN multi-resolution characteristics, without the expensive standard attention.
- The output of dRNNCell is split into “real output”, $\mathbf{y}_t(\mathbf{m}_t)$, which goes to the next layer (cell), and a controlling output, $\mathbf{h}_t$, which is an input to the gating mechanism in the following time steps. In LSTM and GRU, the functions of both outputs are performed by the h -state. In dRNNCell, we split these functions (because they



$$\begin{aligned}
 f_t^i &= \sigma(W_f^i x_t^i + V_f^i h_{t-1}^i + U_f^i h_{t-d}^i + b_f^i) \\
 u_t^i &= \sigma(W_u^i x_t^i + V_u^i h_{t-1}^i + U_u^i h_{t-d}^i + b_u^i) \\
 o_t^i &= \sigma(W_o^i x_t^i + V_o^i h_{t-1}^i + U_o^i h_{t-d}^i + b_o^i) \\
 \tilde{c}_t^i &= \tanh(W_c^i x_t^i + V_c^i h_{t-1}^i + U_c^i h_{t-d}^i + b_c^i) \\
 c_t^i &= u_t^i \otimes (f_t^i \otimes c_{t-1}^i + (1 - f_t^i) \otimes c_{t-d}^i) + (1 - u_t^i) \otimes \tilde{c}_t^i \\
 x_t^1 &= x_t, \quad x_t^2 = x_t \otimes \exp(m_t) \\
 h_{t-1}^1 &= o_t^1 \otimes c_{t-1}^1, \quad h_{t-1}^2 = [h_{t-1,1}^1, \dots, h_{t-1,s_h}^1], \quad m_t = [h_{t,s_h+1}^1, \dots, h_{t,s_h+s_m}^1] \\
 h_{t-d}^1 &= o_t^1 \otimes c_{t-d}^1, \quad h_{t-d}^2 = [h_{t-d,1}^1, \dots, h_{t-d,s_q}^1], \quad y_t = [h_{t,s_q+1}^2, \dots, h_{t,s_q+s_y}^2] \\
 i &= 1 \text{ for bottom cell and } 2 \text{ for upper cell}
 \end{aligned}$$

Fig. 4. adRNNCell.

have different tasks to perform) into two different vectors, thus introducing more flexibility. Another idea behind splitting these functions into two outputs is shown in Ben-Ari and Schwartz-Ziv (2017).

- adRNNCell has an internal attention mechanism that performs as follows. The bottom cell produces attention vector m_t (of the same length as input vector x_t), whose components, after processing by exp function, weights the inputs to the upper cell, x_t^2 . Based on the weighted input vector, x_t^2 , the upper cell produces vector y_t , which feeds the next layer. Note that m_t has a dynamic character. The weights are adjusted by the bottom cell to the current input at step t .

The adRNNCell equations at time t are shown in Fig. 4 where t is a time step; $d > 1$ is a dilation; x_t is an input vector; h_{t-1}^i and h_{t-d}^i are recent and delayed controlling states, respectively; c_{t-1}^i and c_{t-d}^i are recent and delayed cell states, respectively; y_t and m_t are outputs of the upper and bottom dRNNCells, respectively; f_t^i , u_t^i and o_t^i are outputs of the fusion, update and output gates, respectively; $\tilde{c}_t^i$ is a candidate state; W , V and U are weight matrices; b are bias vectors; σ denotes a logistic sigmoid function; and $\otimes$ denotes the Hadamard product.

Fig. 5 shows RNN architecture. It is composed of three layers of adRNNCells dilated 2, 4 and 7, respectively. The number of layers as well as the number of dilations were selected by experimentation and may not be optimal for other forecasting problems. Stacking several layers with hierarchical dilations helps to extract more abstract features

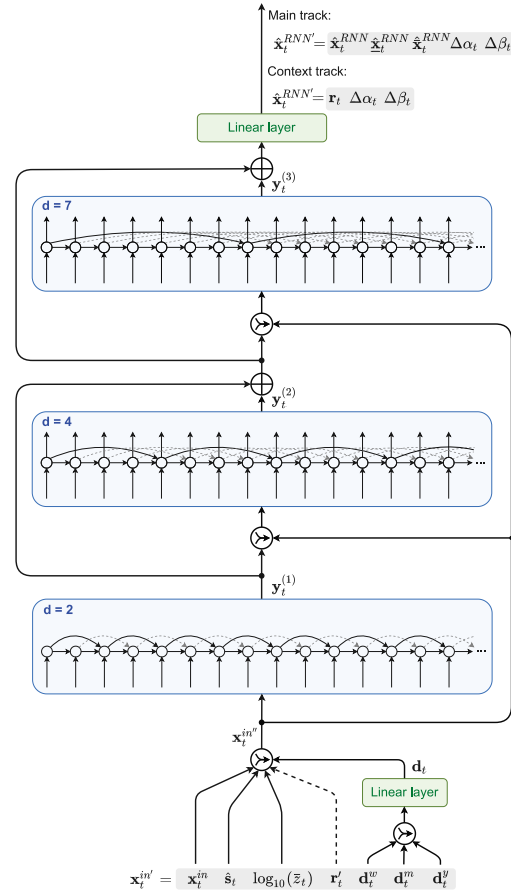


Fig. 5. adRNN architecture (the link shown with a dashed line does not exist in the context track).

in successive layers and obtain a larger receptive field. This makes it easier to learn the long-term and seasonal temporal dependencies of different scales. However, when the number of stacked cells reaches 3 or more, the gradient may vanish, so we use ResNet-style shortcuts between layers (He, Zhang, Ren, & Sun, 2016). Note that input vector $x_t^{in'}$ is introduced not only to the first layer but also to the second and third layers by extending the output vector from the previous layer.

To reduce the input dimensionality of the calendar variables (90 0-1 slots of d_t^w , d_t^m and d_t^y), and meaningfully represent sparse binary vectors, a linear layer embeds them into d -dimensional continuous vectors d_t . Note that the embedding is learned along with the model itself.

Another linear layer, the output one, produces the adRNN output which for the main track is composed of five elements: the vector corresponding to the 24 point forecasts, $\hat{x}_t^{RNN} = [\hat{x}_\tau^{RNN}]_{\tau \in A_t^{out}} \in \mathbb{R}^{24}$, the vector corresponding to the lower bounds of PI, $\hat{x}_t^{RNN} = [\hat{x}_\tau^{RNN}]_{\tau \in A_t^{out}} \in \mathbb{R}^{24}$, the vector corresponding to the upper bounds of PI, $\hat{x}_t^{RNN} = [\hat{x}_\tau^{RNN}]_{\tau \in A_t^{out}} \in \mathbb{R}^{24}$, and corrections for smoothing coefficients, $\Delta \alpha_t$ and $\Delta \beta_t$:

$$\hat{x}_t^{RNN'} = [\hat{x}_t^{RNN}, \hat{x}_t^{RNN}, \hat{x}_t^{RNN}, \Delta \alpha_t, \Delta \beta_t] \quad (8)$$

For the context track, the output linear layer produces three components: the context vector for the i th context series $r_t^{(i)} \in \mathbb{R}^u$, and corrections for smoothing coefficients, $\Delta \alpha_t$ and $\Delta \beta_t$:

$$\hat{x}_t^{RNN'} = [r_t^{(i)}, \Delta \alpha_t, \Delta \beta_t] \quad (9)$$

Please note that the $\Delta \alpha_t$ and $\Delta \beta_t$ in the two equations above, are different, as they are produced by two different RNNs and they are

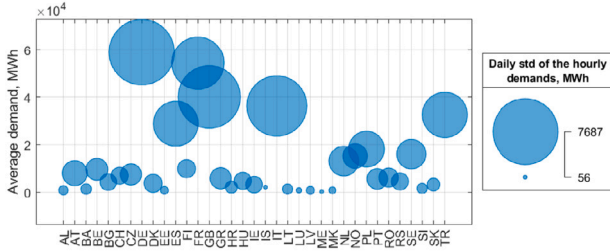


Fig. 6. Average demand and daily dispersion of demands for European countries.

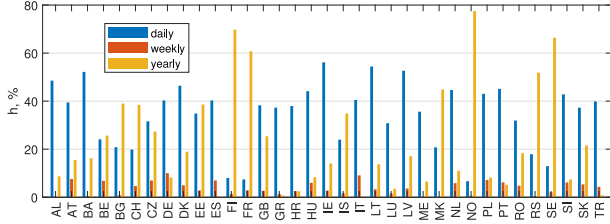


Fig. 7. h -ratio for daily, weekly and yearly harmonics.

per series. For K series in the context batch, context RNN produces K vectors $\mathbf{r}_i^{(j)}$, which are then concatenated, $\mathbf{r}_i = [\mathbf{r}_i^{(1)}, \dots, \mathbf{r}_i^{(K)}] \in \mathbb{R}^{uK}$, and modulated by vectors $\mathbf{g}_i^{(j)}$ (for each series in the main batch, separate j th g -vector is adjusted). They extend the input patterns for the main track, see (4). We expect that the amount of information extracted from each context series to support the forecasting process in the main track is a monotonic function of the size of the individual context vectors, u , although in practice the impact (improvement in accuracy) flattens quickly.

2.7. Modulation of the context vector

Context vector $\mathbf{r}_i$ is the same for every series of the batch in the main track. This is not ideal, although of course due to the complexity of the model and its being recurrent, a different response to the same context information for each series will occur. To make this customization stronger, we apply per time series parameters collected in vectors of the same length as the context vector, $\mathbf{g}_i^{(j)} \in \mathbb{R}^{uK}$, where $j = 1, \dots, b$, and b is the size of the main batch. Modulation vectors $\mathbf{g}_i^{(j)}$ initialized as ones are used to element-wise multiply the context so that each series receives a differently modified context:

$$\mathbf{r}_i'^{(j)} = \mathbf{r}_i \otimes \mathbf{g}_i^{(j)} \quad (10)$$

Modulation vectors $\mathbf{g}_i^{(j)}$ are updated by the same overall optimization procedure (stochastic gradient descent) as RNN weights, with the overarching goal of minimizing the loss function.

2.8. Loss function

The model generates point forecasts as well as the lower and upper bounds of PI. To enable the model to optimize these predictions we employ the following loss function (Smyl et al., 2021):

$$L_\tau = \rho(x_\tau^{\text{out}}, \hat{x}_{q^*, \tau}^{\text{out}}) + \gamma(\rho(x_\tau^{\text{out}}, \hat{x}_{\underline{q}, \tau}^{\text{out}}) + \rho(x_\tau^{\text{out}}, \hat{x}_{\bar{q}, \tau}^{\text{out}})) \quad (11)$$

where

$$\rho(x, \hat{x}_q) = \begin{cases} (x - \hat{x}_q)q & \text{if } x \geq \hat{x}_q \\ (x - \hat{x}_q)(q - 1) & \text{if } x < \hat{x}_q \end{cases} \quad (12)$$

is a pinball loss function defined for x as an actual value, $\hat{x}_q$ as a forecasted value of q th quantile, and $q \in (0, 1)$ as a quantile order; $\tau \in \Delta_\tau^{\text{out}}$; $q^* = 0.5$ corresponds to the median; x_τ^{out} is a normalized actual value (5); $\hat{x}_{q^*, \tau}^{\text{out}} = \exp(\hat{x}_\tau^{\text{RNN}})\hat{s}_{i, \tau}$ is a forecasted value of x_τ^{out} ; $q, \bar{q}$ are the quantile orders for the lower and upper bounds of PI, respectively (e.g. $q = 0.05, \bar{q} = 0.95$); $\hat{x}_{\underline{q}, \tau}^{\text{out}} = \exp(\hat{x}_\tau^{\text{RNN}})\hat{s}_{i, \tau}$ is a forecasted value of $\underline{q}$ -quantile of x_τ^{out} ; $\hat{x}_{\bar{q}, \tau}^{\text{out}} = \exp(\hat{x}_\tau^{\text{RNN}})\hat{s}_{i, \tau}$ is a forecasted value of $\bar{q}$ -quantile of x_τ^{out} ; and $\gamma \geq 0$ is a parameter controlling the impact of the PI component on the loss function (typically between 0.1 and 0.5).

Note that in cross-learning, the loss function evaluates the forecasts of multiple time series having different levels and hence different error ranges. In order to unify the impact of these errors on the learning process, we calculate the loss for the normalized values.

Loss function (11) has three components. The first one, $\rho(x_\tau^{\text{out}}, \hat{x}_{q^*, \tau}^{\text{out}})$, which is symmetric for $q^* = 0.5$, corresponds to the point forecast. The second and third components, $\rho(x_\tau^{\text{out}}, \hat{x}_{\underline{q}, \tau}^{\text{out}})$ and $\rho(x_\tau^{\text{out}}, \hat{x}_{\bar{q}, \tau}^{\text{out}})$, are related to PI bounds. The pinball loss for them is asymmetric to an extent that depends on the quantile orders. Due to the three-component loss function, both point forecasts and PIs can be simultaneously optimized. Hyperparameter γ decides which component is more important in this optimization. For $\gamma = 1$, all components have the same importance. To increase the importance of the first component, we decrease γ towards the limit of 0.

It is worth noting that the pinball loss gives the model the opportunity to reduce the forecast bias by penalizing positive and negative deviations differently. When the model tends to bias positively or negatively, we can reduce this bias by entering q^* less than or greater than 0.5, respectively (see Dudek et al., 2022; Smyl, 2020). In the same way, we can reduce the bias in PI.

3. Experimental study

In this section, we evaluate our proposed model on STLF tasks for 35 European countries. We characterize the data, then we describe the training and optimization procedures, and the baseline models. Next, we present the results, the ablation study and finish by discussing the results and model properties.

3.1. Data

The real-world data was collected from the ENTSO-E repository (www.entsoe.eu/data/power-stats). The dataset comprises hourly electricity demand for 35 European countries (we share this data with the model code in our GitHub repository ([cees-adrnn code and data](https://github.com/cees-adrnn/cees-adrnn), 2022)). The dataset provides a variety of time series with different properties such as the level and trend, stability in variance over time, intensity and regularity of seasonal fluctuations of different periods (annual, weekly and daily), and intensity of random fluctuations. Diversified time series with different properties allow for a more reliable evaluation of the model.

To show the difference in time series, we include two figures. Fig. 6 shows the average demand and daily dispersion of the hourly demands for the countries under consideration in the three-year period 2016–18. The average demand varies from 383 MWh for Montenegro (ME) to 58,941 MWh for Germany (DE), while the daily standard deviation of the hourly demands varies from 56 MWh (Iceland, IS) to 7687 MWh (DE).

Fig. 7 shows the share of the three most significant seasonalities in the time series. This share is expressed by the ratio $h_i = 100A_i^2/(2Var(z_i))$, where A_i is the i th harmonic amplitude (daily, weekly or yearly) and $Var(z_i)$ is the series variance. As can be seen from Fig. 7, for most countries (24) the daily seasonality dominates over others (this is most evident for Bosnia and Herzegovina, Ireland and Lithuania), while for 11 countries the yearly seasonality dominates (Finland, France, Norway and Sweden are in the lead here). The weekly seasonality is less significant with the h -ratio below 10%. Note that

countries similar in terms of average demand and its variance can differ substantially in terms of seasonal components — compare Great Britain (GB) and Italy (IT). In the case of the latter, Fourier analysis does not reveal the yearly seasonality.

3.2. Training, optimization and evaluation setup

The data time span is from 2006 to 2018 but a lot of data is missing in this period. Complete time series, without missing values, were selected to Z_c as context series. They include series for 18 countries: AT, BA, BE, BG, CZ, DE, ES, FR, HR, HU, LU, MK, NL, PL, PT, RO, SI, and SK.

The model development and hyperparameter search were executed by splitting the data into training (years 2006–2015) and validation (2016–2017) sets. Generally, the hyperparameters were chosen so as to minimize the forecasting validation error, however, only after achieving close to zero bias of the forecast, by manipulating q^* — see discussion of the loss function above. Then, using the best hyperparameters, a final training was done on data from 2006 to 2017, and the test was performed on 2018 data. We predict the daily load profiles for each day of the test period, for each of the 35 countries. Due to missing values for Estonia and Italy (the last month of 2018 is missing) and Latvia (the last two months of 2018 are missing), the final errors do not cover these missing periods.

Each training epoch is composed of n_o of “sub-epochs”, defined in the traditional fashion as one scan of all available data. The number of sub-epochs was determined from Smyl et al. (2021):

$$n_o = \min \left(1, \left(\frac{Nb}{L} \right)^p \right) \quad (13)$$

where N is the maximum number of updates per epoch, b is the current batch size, L is the number of time series in the data set, and $p \in [0, 1]$ is a hyperparameter, which by experimentation is set to 0.8.

We use the same schedule of increasing batch sizes and decreasing learning rates proposed in Smith, Kindermans, Ying, and Le (2018). We start with a small batch size of 2, and increase it, although only once, due to the small number of series, to 5 at epoch 4. We use another schedule of decreasing learning rates, which has a similar, but not exactly the same effect as increasing the batch size, this allows the validation error to be further reduced. We use the following schedule: $3 \cdot 10^{-3}$ (epochs 1–5), 10^{-3} (epoch 6), $3 \cdot 10^{-4}$ (epoch 7), and 10^{-4} (epochs 8–9).

The sizes of c -state and h -state were $s_c = 150$, $s_h = 70$. These values were obtained by experimentation starting with $s_c = 50$, $s_h = 20$ and increasing it 3 times, by around 50 and 25, respectively. During each epoch a number of updates is executed, each guided by average error accumulated by executing up to l_o (e.g. 50) forward steps, moving by one day, on a batch. The starting point is chosen randomly and this may cause the number of batch steps be smaller than 50. The batches include random b series. The model is trained using an Adam optimizer.

In Smyl et al. (2022) we discussed the architecture-related hyperparameters, please refer there for more details. As stated there, using experimentation and some rules of thumb (e.g. one of the dilations should be equal seasonality length) we arrive at three blocks, each composed of one cell, with the following dilations: 2, 4 and 7, see Fig. 5.

As described in Section 2 C, the pinball loss function was utilized, with three different quantile values q , to achieve quantile regression for 0.5, 0.05, and 0.95. The actual values for q^* , q , and $\bar{q}$ were slightly different: 0.525, 0.045, 0.975. These values were arrived at by fine-tuning during validation, reducing the bias of the center value, and trying to match the expected percentage of exceedance for 0.05 and 0.95 PIs (5%).

Size of the context batch K was 18. It included all series without any missing values. This condition simplified the code, and as 18 is a relatively small number, all these series could be processed in one batch.

The size of the individual context (size of the output of the context RNN) u was 3 and was chosen by experimentation starting from 1. Generally, it should not be too large, so the $u \cdot K$ should not be much larger than the rest of the combined input into the forecasting RNN. The context is, after all, secondary information.

Finally, we used the ensemble size of $E = 100$, although a size as small as 5 is often sufficient. Simple mean aggregation was applied to the forecasts. For other details see Smyl et al. (2021).

3.3. Baseline models

The performance of the proposed model is compared with the baseline models. They include statistical models, classical ML models, as well as recurrent, deep and hybrid NN architectures:

- Naive — naive model in the form: the forecasted demand profile for day i is the same as the profile for day $i - 7$
- ARIMA — autoregressive integrated moving average model (Dudek, 2015),
- ES — exponential smoothing model (Dudek, 2015),
- Prophet — modular additive regression model with nonlinear trend and seasonal components (Taylor & Letham, 2018),
- N-WE — Nadaraya–Watson estimator (Dudek, 2015)
- GRNN — general regression NN (Dudek, 2016),
- MLP — perceptron with a single hidden layer and sigmoid nonlinearities (Dudek, 2016),
- SVM — linear epsilon insensitive support vector machine (ϵ -SVM) (Peřka, 2021),
- LSTM — long short-term memory (Peřka & Dudek, 2020),
- ANFIS — adaptive neuro-fuzzy inference system (Peřka & Dudek, 2018),
- MTGNN — graph NN for multivariate time series forecasting (Wu et al., 2020b).
- DeepAR — autoregressive RNN model for probabilistic forecasting (Salinas, Flunkert, Gasthaus, & Januschowski, 2020),
- WaveNet — autoregressive deep NN model combining causal filters with dilated convolutions (van den Oord et al., 2016),
- N-BEATS — deep NN with hierarchical doubly residual topology (Oreshkin, Carpio, Chapados, & Bengio, 2020),
- LGBM — Light Gradient-Boosting Machine (Ke et al., 2017),
- XGB — eXtreme Gradient Boosting algorithm (Chen & Guestrin, 2016),
- ES-adRNN100 — hybrid model combining ES and dilated RNN with attention mechanism (Smyl et al., 2022) (predecessor of the proposed model). The result are presented for ensemble of 100 ES-adRNN base models.

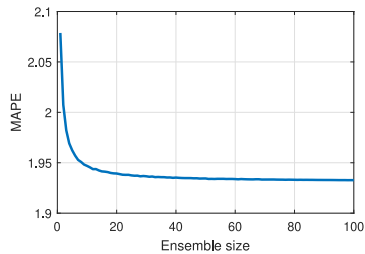
In our recent works Smyl et al. (2022) and Smyl et al. (2021), the models were trained on the data from the period 2016–18. In this work we extend the training period to 2006–18. But some models achieved better results on the shorter period 2016–18, and some of them were not able to learn on data with missing values (about 40% of the countries have incomplete data in the period 2006–2015). For these models, which are marked with an asterisk in Table 2, we provide results for the shorter training period.

3.4. Results

The forecasting quality metrics are shown in Table 2. They include: 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). We provide the results of our proposed model in three variants: an individual model, cES-adRNN, an ensemble of five individual models, cES-adRNN5, and an ensemble of 100 individual models, cES-adRNN100.

Table 2
Quality metrics.

	MAPE	MdAPE	IqrAPE	RMSE	MPE	StdPE	GWtest
Naive*	5.08	4.84	3.32	704.34	−0.26	7.91	1.05
ARIMA*	3.30	3.01	3.00	475.09	−0.01	5.31	18.35
ES*	3.11	2.88	2.73	439.26	0.01	5.13	27.37
Prophet*	4.53	4.32	3.03	619.39	−0.13	6.82	6.77
N-WE*	2.49	2.28	2.30	332.49	−0.13	4.26	59.25
GRNN*	2.48	2.28	2.27	332.91	−0.11	4.25	61.80
MLP	2.92	2.68	2.78	395.72	0.09	5.04	33.68
SVM*	2.55	2.29	2.52	357.24	−0.13	4.37	56.69
LSTM*	2.76	2.57	2.52	381.76	0.02	4.47	39.70
ANFIS*	3.65	3.17	3.66	507.08	−0.10	6.43	13.53
MTGNN	2.87	2.62	2.59	372.65	−0.02	4.64	33.68
DeepAR*	3.42	3.25	2.95	487.14	−0.51	5.16	17.59
WaveNet*	3.03	2.84	2.69	417.49	−0.83	4.68	28.42
N-BEATS*	2.56	2.36	2.39	356.83	−0.04	4.29	51.58
LGBM	2.87	2.60	2.52	391.16	−0.05	4.64	36.99
XGB	2.69	2.43	2.42	366.97	0.00	4.20	48.57
ES-adRNNe100	2.14	1.93	2.09	290.89	−0.11	3.60	81.95
cES-adRNN	2.08	1.86	2.04	281.15	−0.25	3.39	84.66
cES-adRNNe5	1.96	1.74	1.93	265.37	−0.25	3.22	93.23
cES-adRNNe100	1.93	1.71	1.91	262.65	−0.25	3.19	95.49

**Fig. 8.** MAPE depending on the ensemble size.

As can be seen from Table 2, the hybrid models combining ES and adRNN outperform all other models in terms of MAPE, MdAPE and RMSE. Even in its individual version, cES-adRNN, it produces more accurate forecasts than ensembled models such as N-BEATS and boosted-tree models. How ensembling improves the results of cES-adRNN is shown in Fig. 8. For over 20 ensemble members, the MAPE stabilizes, reaching 1.93 for 100 members. The proposed model with a context track is substantially better in terms of accuracy than its predecessor: compare cES-adRNNe100 and ES-adRNNe100 in Table 2. MAPE and RMSE were reduced by almost 10%, while MdAPE was reduced by more than 11%. Fig. 9 shows MAPE for individual countries. For all countries, cES-adRNNe100 yielded the lowest errors except Estonia (EE), where it was beaten by N-BEATS.

Note the lowest values of IqrAPE and StdPE in Table 2 for the proposed model, which indicates the least dispersed predictions compared to the baseline models. The only category in which our model failed to beat other models is MPE. MPE reflects a forecast bias. Note that our model can reduce the bias by selecting the appropriate quantile order for the loss function. But it must be remembered that bias reduction may have a negative impact on the forecast error which is the main quality measure for us. This was the reason we did not reduce the bias further, leaving it at −0.25.

To confirm the best accuracy of the proposed model, a pairwise one-sided Giacomini-White test for conditional predictive ability was performed individually for each country (Giacomini & White, 2006). The results are summarized in Table 2, where GWtest is a percentage of cases for which a model is significantly better in terms of MAPE than other models at $\alpha = 0.05$. GWtest = 95.49 for cES-adRNNe100 means that this model in 95.49% pairwise comparisons with other models had significantly lower MAPE.

Fig. 10 shows examples of forecasts produced by cES-adRNNe100 (weekly period from January 2018). The PIs are also shown in this

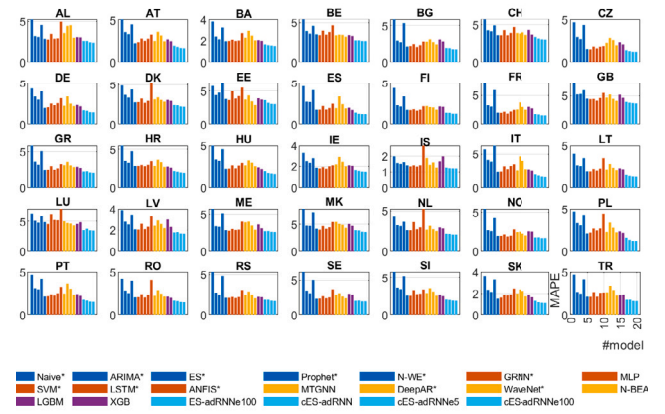
**Fig. 9.** MAPE for individual countries.

figure. To evaluate PIs, we calculate for each country the number of observed values in PIs, below PIs and above PIs. The target values for the assumed quantiles are: 90%, 5% and 5%, respectively. We achieved: $90.51\% \pm 3.21\%$, $5.86\% \pm 1.85\%$ and $3.63\% \pm 1.47\%$, respectively. In addition, we calculate the Winkler score which is defined as the length of the PI plus a penalty if the observation is outside the interval. To make Winkler scores for different countries comparable, we divide each score by the average series value for the country in the test period. The mean normalized Winkler score is $0.1229\% \pm 0.2374\%$.

3.5. Ablation study

In this section, we investigate new elements of cES-adRNNe and estimate their influence on the results. We compare the results of the full model with the results of the reduced model:

- Ab1** ES-adRNNe, without the context track. This is a predecessor of cES-adRNNe described in Smyl et al. (2022). Only the main track is used, so no additional inputs from the context track are introduced.
- Ab2** cES-adRNNe without connections introducing input vector $\mathbf{x}_t^{in'}$ directly to the second and third layers by extending the output vector from the previous layer (see Fig. 5).
- Ab3** cES-adRNNe without adaptive modulation vector $\mathbf{g}$. Thus the context vector is the same for all series in the batch of the main track.
- Ab4** Learning on a shorter period 2016–17. In our previous works on ES-RNN for STLTF we used such a period because it does not contain missing data. In this work, we extend this period to 2006–17.

Other mechanisms and components of cES-adRNNe, which influence its performance were investigated in Smyl et al. (2021) (see Section 4D Ablation Study). They include ES component, ResNet-style shortcut connections, fusion gate, recent and dilated states, selection of the input data, and embedding of the calendar data. It was shown that all of them are beneficial to the performance of the ES-RNN model for STLTF.

Table 3 shows MAPE and RMSE for the full model and its reduced variants (results for the ensemble of 5 base learners are shown). As can be seen from this table, the most beneficial action to improve the accuracy of the model is to train it on longer time series, from 2006 to 2017 (Ab4). In this case, MAPE was reduced from 2.25 to 1.96. The second most beneficial operation was introducing the context track, which supports the learning of the main track (Ab1). This reduced MAPE by 0.18 percentage point. Introducing an adaptive modulation

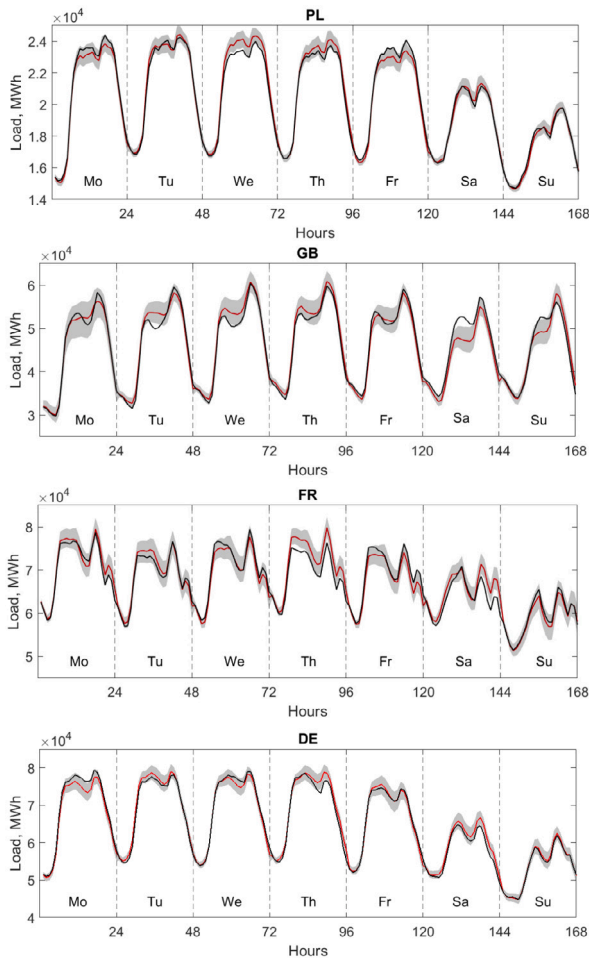


Fig. 10. Examples of forecasts produced by cES-adRNNe100 (real in black, forecasts in red, 90% PIs in gray).

Table 3
Results of ablation study.

	Full	Ab1	Ab2	Ab3	Ab4
MAPE	1.96	2.14	1.98	2.03	2.25
RMSE	265	292	269	276	307

vector adjusted to each series (Ab3) reduced MAPE by 0.07 percentage points. The smallest reduction in error was observed after introducing direct links between the input layer and subsequent recurrent layers (Ab2).

3.6. Discussion

Our proposed model, cES-adRNNe100, outperformed all baseline models in terms of all quality metrics excluding MPE. The best accuracy of our model was confirmed using the Giacomini–White test, which is considered the most appropriate statistical test for comparing forecasts generated by different models.

Context information adjusted by a modulation vector to individual time series improves the forecasting accuracy, as shown by the ablation study (Ab1, Ab3). Even old information, from a few years ago, was important for forecasting new data (Ab4), although it seems intuitive that it should not have such a significant impact on the current dynamics of the series. By extending the training period to 2006–18, the model was able to reduce MAPE and RMSE by more than 12%.

The proposed model is trained globally across all time series (cross-learning) to exploit the information dispersed across many series

(Januschowski et al., 2020). This is a type of multi-task learning (Caruana, 1997) based on latent correlation across multiple series (Smyl, 2020). It improves generalization by preventing overfitting to individual series and, in addition, it significantly speeds up training. Although the model is trained globally, this does not mean that it is not sensitive to the individual properties of the time series. They are expressed in per-series ES parameters and modulation vectors. The characteristic features of each individual series are extracted dynamically by ES and used for learning an appropriate representation for RNN.

RNN, due to its multiple dilated stacked architecture based on a new type of cells, adRNNCells, is able to model complex time series with multiple seasonality. The delayed connections of adRNNCells extend their receptive fields and enable them to model not only short-term but also long-term and seasonal relationships. This is done in a hierarchical manner: subsequent layers expand the receptive field more and more. The attention mechanism built into adRNNCells selects the most relevant input information for improving the forecast accuracy. This mechanism is implemented by a recurrent cell (bottom dRNNCell) to make it dynamic. In each recurrent cycle, a new attention vector is produced based on the current input vector and historical information accumulated in the cell states.

A hybrid architecture enables the model to be optimized and trained as a whole. All the components of the context track and the main track, i.e. ES components, adRNNs, and modulation vector production are optimized simultaneously by the same optimization algorithm. This entails learning series representations in both tracks, learning the context vector by the context adRNN, learning a way to produce modulation vector, learning the smoothing coefficients and initial ES parameters, and learning point forecasts and predictive intervals by the main adRNN at the same time.

The model produces both point forecasts and predictive intervals for them. This is very useful in practical applications because it gives additional information on how reliable the forecast is. To enable the model to predict point forecasts and predictive intervals we introduced the combined loss function based on pinball loss. Using it, the forecast bias can be also controlled.

To prevent overfitting, the model uses ensembling (in the current version we use the simplest variant of ensembling: averaging forecasts generated in the E independent runs of the model with the random initialization, but other more sophisticated ensembling methods are possible, see Dudek et al., 2022; Smyl, 2020). Also, at the time of batch creation, the starting point of training is sampled, and taking into account that the number of steps is limited, each time a series is a member of a batch, a different piece of the series is used. This is an additional mechanism that delays overtraining.

The ResNet-style shortcuts make the system additive: second and third layers add their adjustments to the output of the first layer. Extending input to these two higher layers with $x_t^{in'}$ improves the forecasting performance. Somehow, these additions are more accurate when the higher layers “see” the original input.

The amount of calculations is linearly dependent on the number of parameters, that, generally speaking, grow squarely with the size of the cells’ state, and linearly with the number of layers. Additionally, the amount of calculations grows with the number of steps an RNN takes during training and the batch size. We execute around 10 K updates during training.

One primary limitation of our model is related to the number of time series it can handle effectively. Since the model uses per time series parameters, it works well on datasets with small numbers of series. As each series has to be “seen” many times during training, we speculate that no more than 1 K series could be allowed, given the number of updates the model undergoes during training (which is on the order of 10 K).

4. Conclusion

In this study, we propose a novel contextually enhanced hybrid model for STLTF combining RNN with attentive dilated cells and exponential smoothing. The context track extracts information from the representative time series and introduces it as an additional input to the main track, which produces point forecasts and predictive intervals. The experimental study showed that the context track improves significantly forecasting accuracy. The proposed model outperformed in terms of accuracy all baseline models including statistical models, state-of-the-art machine learning models, and its predecessor, which does not have a context track.

The proposed model is equipped with many mechanisms and procedures designed to increase forecasting efficiency:

- hybrid architecture — ES for on-the-fly preprocessing (learning of time series representation) and RNN for modeling temporal dependencies in the series,
- a context RNN that provides additional global information to the main RNN,
- new recurrent cells with dilation and attention mechanisms, which help in modeling long-term and seasonal dependencies as well as selecting input information,
- dynamic ES model — RNN corrects ES parameters in each recurrent cycle,
- cross-learning, which enables the model to capture shared features of individual series and prevents overfitting,
- quantile loss function which enables the model to produce both point forecasts and predictive intervals, and also to reduce the forecast bias,
- ensembling and mechanisms to delay overfitting.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Grzegorz Dudek reports a relationship with Czestochowa University of Technology that includes: employment. Pawel Pelka reports a relationship with Czestochowa University of Technology that includes: employment. Slawek Smyl reports a relationship with Meta Platforms Inc that includes: employment. Slawek Smyl reports a relationship with Uber Technologies Inc that includes: employment.

Data availability

Data will be made available on request.

Acknowledgments

This paper was supported by grant 020/RID/2018/19 from the Polish Minister of Science and Higher Education titled “Regional Initiative of Excellence”, 2019–23.

Appendix

A.1. Hyperparameters of *cES-admne*

- Batch size (b) — set to 2 and increased to 5 in epoch 4, selected by experimentation. Due to small size of the dataset, it was not feasible to grow it further.
- Dilations — 2, 4 and 7, set by experimentation, but also a heuristic that they should be growing, preferably exponentially, and if possible, some of them should be equal to the seasonality, 7 (days) in our case, as we step by 24 h.

- Embedding size of the calendar variables — set to 4 by experimentation.
- Ensemble combining method — simple mean aggregation.
- Ensemble size (E) — set to 5 or 100.
- Initial smoothing coefficients — $I\alpha = -3.5$, $I\beta = 0.3$; selected by observing average values of the dynamic version of the coefficients.
- Learning rate — set to $3 \cdot 10^{-3}$ in epochs 1–5, 10^{-3} in epoch 6, $3 \cdot 10^{-4}$ in epoch 7, and 10^{-4} in epochs 8–9; selected by experimentation.
- Lengths of warm-up periods — set to $3 \cdot 7$ steps for training and $13 \cdot 7$ steps for testing. They need to be larger than the length of the longest dilation (7) to “warm up” the RNN cell states, and the data was long enough to try these much larger values.
- Number of batch training steps (l_o) — set to 50 by experimentation.
- Number of epochs — set to 10. The program follows a schedule of first increasing batch sizes, and then, in subsequent epochs, decreasing the learning rate. After both schedules finish, we stop.
- Number of sub-epochs (n_0) — determined from (13).
- Number of updates per epochs (N) — set to 2500 by experimentation. The idea was to see a noticeable change in accuracy with each epoch.
- Optimizer — Adam, selected by experimentation.
- Parameter for the number of sub-epochs calculation (p) — set to 0.8, selected from $[0, 1]$ by experimentation. Partially related to epochs’ processing times — we tried to keep it similar across epochs.
- Parameter for the PIs in the loss function (λ) — set to 0.3 based on the rule of thumb that the average training central loss should be larger than those of lower and upper intervals, but not larger than 10 times.
- Pinball loss quantiles — $q^* = 0.525$, $q = 0.045$, and $\bar{q} = 0.975$, set by experimentation, being distortions of the quantiles we have been fitting: 0.5, 0.05, and 0.95, respectively.
- Size of the context batch (K) — set to 18 as the number of series without missing values.
- Size of the c -state (s_c) — set to 150 by experimentation.
- Size of the h -state (s_h) — set to 70 by experimentation.
- Size of output vector y (s_y) — it is the difference between s_c and s_h .
- Size of the context RNN output (u) — set to 3 by experimentation.
- Initial seasonality factors in ES — calculated as a ratio of values from the first input window of the batch and average values of the window, so around 1, appropriate for the multiplicative seasonality mechanism we are using.

A.2. Parameters of *cES-admne*

- Weights and biases of adRNNCells for the main track — there are twelve matrix sets (four for each of the three layers): $\mathbf{W} \in \mathbb{R}^{n \times 150}$, $\mathbf{V} \in \mathbb{R}^{70 \times 150}$, $\mathbf{U} \in \mathbb{R}^{70 \times 150}$, $\mathbf{b} \in \mathbb{R}^{150}$, where $n = 193$ for the first layer and $n = 273$ for the second and third layers; weights and biases are initialized randomly.
- Weights and biases of adRNNCells for the context track — there are twelve matrix sets: $\mathbf{W} \in \mathbb{R}^{m \times 150}$, $\mathbf{V} \in \mathbb{R}^{70 \times 150}$, $\mathbf{U} \in \mathbb{R}^{70 \times 150}$, $\mathbf{b} \in \mathbb{R}^{150}$, where $m = 247$ for the first layer and $m = 327$ for the second and third layers; weights and biases are initialized randomly.
- Weights and biases of the linear layer for embedding calendar variables, used by both the main and context tracks — $\mathbf{W} \in \mathbb{R}^{90 \times 4}$, $\mathbf{b} \in \mathbb{R}^4$; initialized randomly.
- Weights and biases of the output linear layer for the main track — $\mathbf{W} \in \mathbb{R}^{80 \times 74}$, $\mathbf{b} \in \mathbb{R}^{74}$; initialized randomly.

- Weights and biases of the output linear layer for the context track — $\mathbf{W} \in \mathbb{R}^{80 \times 5}$, $\mathbf{b} \in \mathbb{R}^5$; initialized randomly. Value 5 is a sum of the size of the output of the context RNN ($u = 3$) and two outputs used for calculating dynamic smoothing coefficients.
- Matrix of the modulation vectors — $\mathbf{G} \in \mathbb{R}^{b \times 54}$, where b is the size of the main batch (2 or 5); initialized as ones. This matrix is dynamically created for each batch, stacked from the respective per-series parameters, which are vectors.
- There are also two per series initial smoothing parameters.

A.3. Hyperparameters of the baseline models

- ARIMA — time series were decomposed into 24 series (a separate series for each hour) and predicted using `auto.arima` function in R, which automatically selects the best ARIMA model.
- ES — time series were decomposed into 24 series (a separate series for each hour) and predicted using `ets` function in R, which automatically selects the best ES model.
- Prophet — `prophet` function in R was used with default hyperparameters.
- N-WE — our implementation in Matlab. Pattern length was set to 24, and smoothing parameters were selected for each series in cross-validation.
- GRNN — our implementation in Matlab. Pattern length was set to 24, and smoothing parameters were selected for each series in cross-validation.
- MLP — `feedforwardnet` function in Matlab was used. A single hidden layer was used, with the number of hidden nodes selected for each series in cross-validation. A Levenberg–Marquardt algorithm with Bayesian regularization was used for training.
- SVM — `fitrsvm` function in Matlab was used. The hyperparameter values, `BoxConstraint`, `KernelScale` and `Epsilon`, were selected in the automatic optimization procedure implemented in `fitrsvm`.
- LSTM — `lstm` function in Matlab was used. The number of neurons was set to 24 by experimentation. The remaining hyperparameters were set to their default values.
- ANFIS — `anfis` function in Matlab was used. A Sugeno fuzzy inference system was used, with the number of fuzzy rules selected for each series in cross-validation.
- MTGNN — we based on the repository shared by the authors on <https://github.com/nanzhan/MTGNN>. We used a default set of hyperparameters.
- DeepAR — we utilized the GluonTS implementation of DeepAR with MXNet (<https://ts.gluon.ai/stable>). The `context_length` was set to $7 \cdot \text{prediction_length}$. The remaining hyperparameters were set to their default values.
- WaveNet — we utilized the GluonTS implementation of WaveNet with MXNet. The hyperparameters were set to their default values.
- N-BEATS — we utilized the GluonTS implementation of N-BEATS with MXNet. The `context_length` was set to $7 \cdot \text{prediction_length}$, and `meta_loss_function` to `sMAPE`. The remaining hyperparameters were set to their default values.
- LGBM — we used `LGBMRegressor` function from `lightgbm` library. We set `max_depth` to 10, `base_score` to 0.5, `num_iterations` to 1000, `early_stopping_rounds` to 50, `learning_rate` to 0.01, and we added the following features: `dayofyear`, `hour`, `dayofweek`, `quarter`, `month`, `year`, `lag1` — `lag21`. The remaining hyperparameters were set to their default values.
- XGB — we used `XGBRegressor` function from `xgboost` library. We set `max_depth` to 10, `base_score` to 0.5, `booster` to `gbtree`, `n_estimators` to 1000, `objective` to `squared-error`, `early_stopping_rounds` to 50, `learning_rate` to 0.01, and we added the following features: `dayofyear`,

`hour`, `dayofweek`, `quarter`, `month`, `year`, `lag1` — `lag21`. The remaining hyperparameters were set to their default values.

- ES-adRNN100 — our implementation in Python (available as open-source code in the github repository: <https://github.com/slawecki17/ES-adRNN>). The hyperparameters are described in detail in Smyl et al. (2022).

References

- Aprillia, H., Yang, H.-T., & Huang, C.-M. (2020). Statistical load forecasting using optimal quantile regression random forest and risk assessment index. *IEEE Transactions on Smart Grid*, 12(2), 1467–1480.
- Arora, S., & Taylor, J. W. (2018). Rule-based autoregressive moving average models for forecasting load on special days: A case study for France. *European Journal of Operational Research*, 266, 259–268.
- Ben-Ari, I., & Shwartz-Ziv, R. (2017). Sequence modeling using a memory controller extension for LSTM. In *NIPS 2017 time series workshop*.
- Bojer, C. S., & Meldgaard, J. P. (2021). Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2), 587–603.
- Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1), 41–75.
- cES-adRNN code and data: (2022). <https://github.com/slawecki17/ES-adRNN-with-Context>.
- Charlton, N., & Singleton, C. (2014). A refined parametric model for short term load forecasting. *International Journal of Forecasting*, 30(2), 364–368.
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proc. ACM SIGKDD*.
- Chen, K., et al. (2018). Short-term load forecasting with deep residual networks. *IEEE Transactions on Smart Grid*, 10(4), 3943–3952.
- Cho, K., et al. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
- De Stefani, J., & Bontempi, G. (2021). Factor-based framework for multivariate and multi-step-ahead forecasting of large scale time series. *Frontiers in Big Data*, 75.
- Dudek, G. (2015). Pattern similarity-based methods for short-term load forecasting – part 2: Models. *Applied Soft Computing*, 36, 422–441.
- Dudek, G. (2016). Neural networks for pattern-based short-term load forecasting: A comparative study. *Neurocomputing*, 205, 64–74.
- Dudek, G., Pełka, P., & Smyl, S. (2022). A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting. *IEEE Transactions on Neural Networks Learning Systems*, 33(7), 2879–2891.
- Eskandari, H., Imani, M., & Moghaddam, M. P. (2021). Convolutional and recurrent neural network based model for short-term load forecasting. *Electric Power Systems Research*, 195, Article 107173.
- Gao, R., Du, L., Suganthan, P. N., Zhou, Q., & Yuen, K. F. (2022). Random vector functional link neural network based ensemble deep learning for short-term load forecasting. *Expert Systems with Applications*, 206, Article 117784.
- Giacomini, R., & White, H. (2006). Tests of conditional predictive ability. *Econometrica*, 74(6), 1545–1578.
- Granger, C. W. J. (1969). Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3), 424–438.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proc. CVPR* (pp. 770–778).
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
- Hong, Y., Zhou, Y., Li, Q., Xu, W., & Zheng, X. (2020). A deep learning method for short-term residential load forecasting in smart grid. *IEEE Access*, 8, 55785–55797.
- Januschowski, T., et al. (2020). Criteria for classifying forecasting methods. *International Journal of Forecasting*, 36, 167–177.
- Kandilogiannakis, G., Mastorocostas, P., & Voulodimos, A. (2022). ReNFuzz-LF: A recurrent neurofuzzy system for short-term load forecasting. *Energies*, 15(10), 3637.
- Ke, G., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Proc. NIPS*.
- Kim, J., Moon, J., Hwang, E., & Kang, P. (2019). Recurrent inception convolution neural network for multi short-term load forecasting. *Energy and Buildings*, 194, 328–341.
- Kong, W., et al. (2017). Short-term residential load forecasting based on LSTM recurrent neural network. *IEEE Transactions on Smart Grid*, 10(1), 841–851.
- Li, W., Shi, Q., Sibtain, M., Li, D., & Mbanze, D. E. (2020). A hybrid forecasting model for short-term power load based on sample entropy, two-phase decomposition and whale algorithm optimized support vector regression. *IEEE Access*, 8, 166907–166921.
- Li, R., Wang, S., Zhu, F., & Huang, J. (2018). Adaptive graph convolutional neural networks. In *Proc. AAAI*, vol. 32, no. 1.
- Lin, W., & Wu, D. (2021). Residential electric load forecasting via attentive transfer of graph neural networks. In *IJCAI* (pp. 2716–2722).
- Massaoudi, M., et al. (2021). A novel stacked generalization ensemble-based hybrid LGBM-XGB-MLP model for short-term load forecasting. *Energy*, 214, Article 118874.
- Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *Proc. ICLR*.
- Pełka, P. (2021). Pattern-based forecasting of monthly electricity demand using support vector machine. In *Proc. IJCNN* (pp. 1–8).

- Pełka, P., & Dudek, G. (2018). Neuro-fuzzy system for medium-term electric energy demand forecasting. In *Proc. ISAT* (pp. 38–47). Springer AISC 655.
- Pełka, P., & Dudek, G. (2020). Pattern-based long short-term memory for mid-term electrical load forecasting. In *Proc. IJCNN* (pp. 1–8).
- Piotrowski, P., Baczyński, D., Kopyt, M., & Gulczyński, T. (2022). Advanced ensemble methods using machine learning and deep learning for one-day-ahead forecasts of electric energy production in wind farms. *Energies*, 15(4), 1252.
- Qin, Y., et al. (2017). A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*.
- Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3), 1181–1191.
- Satorras, V. G., Rangapuram, S. S., & Januschowski, T. (2022). Multivariate time series forecasting with latent graph inference. *arXiv preprint arXiv:2203.03423*.
- Sen, R., Yu, H.-F., & Dhillon, I. S. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. 32, In *Proc. NeurIPS*, 2019.
- Shi, H., Xu, M., & Li, R. (2017). Deep learning for household load forecasting—A novel pooling deep RNN. *IEEE Transactions on Smart Grid*, 9(5), 5271–5280.
- Smith, S. L., Kindermans, P. J., Ying, C., & Le, Q. V. (2018). Don't decay the learning rate, increase the batch size. In *Proc. ICLR*.
- Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1), 75–85.
- Smyl, S., Dudek, G., & Pełka, P. (2021). ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting. *arXiv preprint arXiv:2112.02663*.
- Smyl, S., Dudek, G., & Pełka, P. (2022). ES-dRNN with dynamic attention for short-term load forecasting. In *Proc. IJCNN* (pp. 1–8).
- Takeda, H., Tamura, Y., & Sato, S. (2016). Using the ensemble Kalman filter for electricity load forecasting and analysis. *Energy*, 104, 184–198.
- Tan, M., et al. (2019). Ultra-short-term industrial power demand forecasting using LSTM based hybrid ensemble learning. *IEEE Transactions on Power Systems*, 35(4), 2937–2948.
- Taylor, J. W. (2012). Short-term load forecasting with exponentially weighted methods. *IEEE Transactions on Power Systems*, 27(1), 458–464.
- Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37–45.
- Tran, A., Mathews, A., Ong, C. S., & Xie, L. (2021). Radflow: A recurrent, aggregated, and decomposable model for networks of time series. In *Proc. WWW* (pp. 730–742).
- van den Oord, A., et al. (2016). WaveNet: A generative model for raw audio. *arXiv: 1609.03499*.
- Wang, S., Wang, X., Wang, S., & Wang, D. (2019). Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting. *International Journal of Electrical Power & Energy Systems*, 109, 470–479.
- Wang, Y., et al. (2019). Deep factors for forecasting. In *Proc. ICML* (pp. 6607–6617).
- Wang, Y., et al. (2020). Short-term load forecasting for industrial customers based on TCN-lightgbm. *IEEE Transactions on Power Systems*, 36(3), 1984–1997.
- Wu, Z., et al. (2020a). Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proc. 26th ACM SIGKDD* (pp. 753–763).
- Wu, Z., et al. (2020b). Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proc. ACM SIGKDD*.
- Yang, D., Guo, J., Sun, S., Han, J., & Wang, S. (2022). An interval decomposition-ensemble approach with data-characteristic-driven reconstruction for short-term load forecasting. *Applied Energy*, 306, Article 117992.
- Zhou, J., et al. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1, 57–81.



Full length article

Any-quantile probabilistic forecasting of short-term electricity demand: Fusing uncertainties from diverse sources

Slawek Smyl^a, Boris N. Oreshkin^b, Paweł Pełka^c, Grzegorz Dudek^c,*

^a Walmart, USA

^b Amazon, Canada

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

ARTICLE INFO

Dataset link: www.entsoe.eu/data/power-stats,
<https://github.com/boreshkinai/any-quantile>

Keywords:

Short-term load forecasting
Probabilistic forecasting
Quantile forecasting
Multi-source uncertainty fusion
Cross-learning
Deep learning

ABSTRACT

Power systems operate under significant uncertainty arising from diverse and dynamic factors such as fluctuating renewable energy generation, evolving consumption patterns, and complex market dynamics. Accurately forecasting electricity demand necessitates advanced methodologies capable of capturing these multifaceted uncertainties. Our work develops any-quantile probabilistic forecasting framework, which enables the generation of forecasts for arbitrary quantile levels at inference time using a single trained model. This constitutes a substantial methodological advancement over traditional quantile regression techniques, which typically require training a separate model for each quantile or limiting predictions to a fixed set of predefined quantile levels. We show that integrating this framework into state-of-the-art neural architectures, specifically ESRNN and N-BEATS, yields superior distributional forecasting performance in the context of short-term electricity demand. Additionally, we develop the general Bayesian theory of cross-learning and link its latent objects with the elements of our architectures, providing a Fusion theory foundation for cross-learning from multiple power systems.

Empirical validation utilizing a comprehensive dataset of hourly electricity demand from 35 European countries showcases the efficacy of our approach, demonstrating superior predictive performance and enhanced quantile forecasting accuracy.

1. Introduction

Extraneous stochastic factors, which include weather fluctuations, load variations and equipment failures, make probabilistic forecasting arguably the only viable approach to forecasting in power systems [1]. Concretely, these factors introduce uncontrollable random variability in power system operations, which can only be accounted for, at any particular point in time, by keeping a model of many possible futures using the tools of probabilistic calculus. Probabilistic calculus induces probabilistic forecasts that provide likely distribution of electricity demand for a specific point in the future or a set of likely demand trajectories in the context of multi-horizon forecasting. Power system operators can use these probabilistic forecasts to enhance operational efficiency, mitigate risks and ensure reliable and secure power supply. This correlates with the following range of the applications of forecasting in power systems: (i) planning and optimization, (ii) security assessment and management and (iii) energy trading. In the following, we discuss the motivations behind the use of probabilistic forecasting in these application areas in more detail.

Planning and optimization benefit from probabilistic forecasting by incorporating probabilistic forecasts of load demand and generation availability to minimize operating costs, maximize revenue and/or ensure reliable power supply via stochastic optimization and scheduling. For example, operating reserve can be planned for uncertainties to maintain grid reliability during peak demand periods, unexpected events or sudden changes in weather conditions. Concretely, the integration of probabilistic forecast information can reduce the predicted regulation range by about 12%–31% [2], which reduces the necessary reserve margin costs and helps better utilize limited system balancing resources. Wang et al. [3] show that the probabilistic forecast information can be transformed into dynamic reserve requirements, which is beneficial in terms of reducing the cost and the number of power balance and reserve violations compared to the static reserve requirements. Finally, probabilistic renewable energy generation forecasts enable the design of ramping products that provide flexible and efficient resources to balance the grid during periods of changing

* Corresponding author.

E-mail addresses: slaweks@hotmail.co.uk (S. Smyl), boris.oreshkin@gmail.com (B.N. Oreshkin), pawel.pelka@pcz.pl (P. Pełka), grzegorz.dudek@pcz.pl (G. Dudek).

<https://doi.org/10.1016/j.inffus.2025.103637>

Received 3 April 2025; Received in revised form 16 July 2025; Accepted 14 August 2025

Available online 23 September 2025

1566-2535/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC license (<http://creativecommons.org/licenses/by-nc/4.0/>).

generation patterns, ensuring the seamless integration of renewable energy into the power system [4].

Security assessment and management in power systems focus on identifying high-risk extreme events and applying preventive measures accordingly [5]. For example, probabilistic forecasts of electricity demand can be used to identify high-likelihood future load demand increase events and operators can take preventive measures such as activating additional generation resources, adjusting generation schedules, or implementing load shedding plans to ensure grid stability. Other stability risks such as voltage fluctuations, frequency deviations and power imbalances, can also be better managed using probabilistic forecasts.

Energy trading optimizes generation and consumption patterns, manages risks and maximizes economic benefits by buying and selling electricity in various markets, such as wholesale electricity markets, balancing markets and ancillary services markets [6]. Probabilistic forecasting provides valuable insights to energy traders for optimizing their trading strategies. It quantifies the range of possible outcomes and associated uncertainties in key parameters, such as electricity prices, load demand and generation availability, which are essential for trading decisions. For example, if a probabilistic load forecast indicates a higher probability of a peak load scenario, traders can adjust their generation schedules or contract additional capacity to meet the expected demand.

In our current work, we focus on the short-term electricity demand forecasting, which has prominent applications in energy trading as well as short-term planning and scheduling optimization. The rest of the section provides the review of the most relevant literature in probabilistic forecasting and its applications in power systems, as well as summarizes core motivation and contributions of our work.

1.1. Related work

Probabilistic forecasting in power systems has profound impact on operational security, results in quantifiable financial outcomes and as such has profound impact on decision making in energy sector as a whole. The concept of probabilistic load forecasting is comprehensively examined in [7], focusing on the progress in general methodology and the impact in practical applications. Practical applications encompass such areas as probabilistic load flow, unit commitment and reliability planning. In contrast to point forecasting, probabilistic load forecasting characterizes load variations by providing predictions in the form of probability density functions (PDFs), quantiles, or prediction intervals (PIs). Practical examples of the use of forecast uncertainty in grid management are illustrated in [1]. These include the employment of probabilistic data for grid security calculations, the optimization of unit commitment, the mitigation of risks associated with extreme conditions and events. Furthermore, the review of probabilistic forecasting in smart grid applications can be found in [8], while applications of probabilistic forecasting in solar power systems are reviewed in [9].

Probabilistic forecasting approaches, as delineated by Hong et al. [7], can be broadly subdivided into three primary categories: (i) input scenario simulation, (ii) post-processing involving residual simulation or forecast combination and (iii) model-dependent interval construction and probabilistic forecasting models. In the first category, multiple scenarios for input variables, such as temperature, are employed to generate various point forecasts. These forecasts are then aggregated to create a probabilistic forecast. For example, [10] employs semi-parametric additive models to estimate the relationships between demand and various driver variables, including temperatures, calendar effects and some demographic and economic variables. Subsequently, the distributions of annual and weekly peak electricity demands are forecasted by utilizing a mixture of temperature simulations, anticipated future economic scenarios and residual bootstrapping. In another approach, Taylor and Buizza [11] use neural networks (NNs) for estimating probabilistic load

forecasts, with weather ensemble predictions for multiple scenarios being integrated as input variables. Additionally, [12] introduces generic scenario-based probabilistic load forecasting models using an ensemble of regression trees. In this case, quasi-periodic time-series data are represented as matrices and the singular value decomposition is employed to generate temperature scenarios. Finally, [13] explores and empirically compares three basic methods for generating temperature scenarios: the fixed-date, shifted-date and bootstrap.

The second approach to generating probabilistic forecasts involves post-processing existing point forecasts. This is typically accomplished by applying the probability density function of residuals on top of existing point forecasts. For example, a common approach consists in fitting a normal distribution to forecasting errors [14]. The key takeaway of the study is that the quality of the underlying model plays a crucial role in ensuring the success of post-processing probabilistic load forecast. The literature also includes more advanced methods of residual simulation, such as the application of GARCH [15]. Moreover, residual simulation can be successfully combined with predicted model uncertainties as shown in [16]. This requires a more sophisticated modeling approach employing a hybrid ensemble of deep belief networks, bagging, boosting approaches and a k -nearest neighbor classifier. Similarly, ensemble forecasting can be used as a standalone approach to generating probabilistic forecasts: a set of point forecasts produced by individual models can be transformed into a PI via a quantile regression as shown in [17].

The third probabilistic forecasting category encompasses forecasting models capable of generating probabilistic forecasts directly. Bayesian non-parametric techniques such as quantile regression based on Gaussian process are popular in this category. For example, in [18] Gaussian process predicts load quantiles in the next time step and the probability density function is obtained through kernel density estimation (KDE). Similar forecasting approaches, involving feature selection, quantile modeling and KDE are employed in [19–21]. The lasso method acts as feature selector, while neural network models quantiles via quantile regression in [19]. [20] utilizes a copula model for feature selection, while combining a 1D Convolutional Neural Network (CNN) and a Gated Recurrent Unit (GRU) to produce quantile forecasts. In [21], quantile forecasts produced by different quantile regression neural models are transformed into PDFs through KDE. These PDFs are subsequently combined using a multilayer Gaussian mixture distribution. Li et al. [22] extend [20] by additionally incorporating a time attention mechanism. This model additionally allows for quantitative modeling of two types of uncertainty: epistemic and aleatoric, which is achieved through the use of Monte Carlo dropout and an improved loss function based on the Kullback–Leibler divergence. Similar KDE-based approaches include [23–25]. He et al. [23] propose a kernel-based support vector quantile regression model using Copula theory, which can quantify the uncertainty between inputs, such as power load and real electricity price. The forecasting results under different quantiles are fed into the KDE function. KDE can also be used on top of basic learners as a part of the meta-learning approach [24]. Here, a quantile regression LSTM meta-learner combines point forecasts produced by tree-based models, including random forest, gradient boosting decision tree and light gradient boosting machine. The PDF is determined using KDE modified by Gaussian approximation of quantiles. Similarly, a quantile meta-model, in the form of quantile linear regression or quantile random forest, can be constructed by stacking on top of the array of base models as shown in [25].

Parametric and non-parametric probabilistic forecasting. In addition to the mechanism used to construct the forecast, probabilistic forecasting approaches can be categorized into parametric, semi-parametric and non-parametric [26]. All of the models discussed thus far are representatives of the non-parametric family. Further examples of non-parametric methods are provided below. Their common trait is the use of advanced machine learning models in combination with the variants of pinball loss for supervision. For example, [27] employs

a Long Short-Term Memory (LSTM) network, [28] uses feed-forward network and [29] combines GRU and temporal fusion transformer, while all of them are supervised with pinball loss. Additionally, [29] adds quantile constraints and PI penalty terms to prevent quantile crossover and aid in constructing tighter PIs. Smyl et al. [30] combine exponential smoothing with a recurrent neural network (RNN) based on a novel recurrent cell, specifically designed to efficiently model both short and long-term dependencies within time-series data. The resulting probabilistic forecasts are presented in the form of three quantiles, corresponding to the median and the bounds of PI. Alfieri and De Falco [31] decompose the load time-series through wavelet decomposition and predict its components using quantile regression forest. The final load forecasts are reconstructed by interpolating quantile predictions to prevent quantile crossing.

A parametric approach to generating probabilistic forecasts assumes a certain PDF for the forecast distribution and involves a forecasting model that produces parameters of this PDF. In its simplest form, PDF can be represented by a normal distribution with mean and variance parameters. Gaussian processes are a viable option for slightly more general modeling [32]. A more heavy-tail model such as Student-t distribution can be used on top of MLP, DeepAR and Transformer models, as done in GluonTS library [33]. Another approach to handling multi-modality and asymmetry of forecast distribution is based on Gaussian mixture model [34].

1.2. Motivation and contributions

It is crucial to acknowledge the growing array of factors contributing to significant uncertainty in the energy sector. These encompass the dynamic expansion of distributed renewable energy sources, the evolution of smart grids, the advancement of energy storage systems, the increasing prevalence of electric vehicles, the adoption of extended demand response programs and the liberalized markets with increasingly intricate pricing policies. These factors must contend with sudden weather changes and extreme weather events, further complicating the landscape. The fusion of information from these diverse and dynamic sources of uncertainty is inherently complex, necessitating advanced methodologies to effectively manage and mitigate the resultant unpredictability in energy systems.

The value of probabilistic forecasting in the context of tackling these challenges in power systems applications has been clearly demonstrated in the literature. At the same time, there is a clear gap in the probabilistic forecasting accuracy, which is primarily related to (i) the prevalent use of simple non-state-of-the-art architectural backbones (LSTM/GRU/CNN) that lag behind in terms of point forecasting accuracy and (ii) the prevalent use of simple quantile regression training and inference mechanisms that lag behind in terms of probabilistic forecasting accuracy.

In this paper, we address the existing gap by demonstrating that integrating state-of-the-art neural architectures, ESRNN (Exponential Smoothing Recurrent Neural Network) and N-BEATS (Neural Basis Expansion Analysis for Time Series), with advanced any-quantile distributional training, alongside cross-learning information fusion from diverse sources, significantly enhances both point and probabilistic forecasting accuracies.

We selected these two architectures for modification because, as their original authors (ESRNN was proposed by Smyl in [35], and N-BEATS was introduced by Oreshkin et al. in [36]), we possess deep insight into their design and potential. Both architectures have demonstrated strong performance in various challenging forecasting tasks – most notably, ESRNN won the prestigious M4 forecasting competition – and have proven effective in the context of electricity demand forecasting [30,37].

In a nutshell, our research contributions can be summarized as follows:

1. We introduce a novel any-quantile probabilistic forecasting framework that enables the generation of arbitrary quantiles at inference time using a single trained model. The approach is model-agnostic and can be applied to various machine learning architectures.
2. We develop a Bayesian theory of cross-learning and establish its correspondence with architectural components, laying a theoretical foundation for fusion-based learning across multiple power systems.
3. We propose architectural modifications to the ESRNN forecasting algorithm expanding it with the any-quantile capability.
4. We propose architectural modifications to the N-BEATS forecasting algorithm expanding it with the any-quantile capability.

In the experimental section, we present empirical results for the short-term electricity demand forecasting task, validating the proposed contributions. The results demonstrate that the any-quantile-enabled ESRNN and N-BEATS models consistently outperform a range of baseline methods, including statistical approaches and state-of-the-art machine learning models, primarily neural forecasting architectures.

The rest of the paper is organized as follows. Section 4 formulates the problem and conveys analysis results. Section 5 describes proposed neural architectures solving any-quantile forecasting task. Section 6 describes our empirical findings. Finally, Sections 7 and 8 discuss our findings and conclude the paper.

2. Short-term distributional forecasting problem

We define the forecasting problem with a forecast horizon of length H and a historical time-series $[y_1, \dots, y_T] \in \mathbb{R}^T$, where T is the time stamp of the last observation. Forecaster function $f_\theta : \mathbb{R}^{W \times 1} \rightarrow \mathbb{R}^H$, parameterized with $\theta \in \Theta$, takes the *lookback window* $\mathbf{x} \in \mathbb{R}^W = [y_{T-W+1}, \dots, y_T]$ of length $W \leq T$ ending with the last observed value y_T , along with quantile q and predicts the vector of q th quantiles of future values $\hat{\mathbf{y}}_{T,q} \in \mathbb{R}^H = [y_{T+1,q}, \dots, y_{T+H,q}]$ based on past observations. The indices of this window belong to the index set $\Delta_T^n = \{T - W + 1, \dots, T\}$. In the following, the forecast of the q th quantiles, $\hat{\mathbf{y}}_{T,q}$, is denoted as $\hat{\mathbf{y}}_q$, dropping time index T for brevity, unless specifically stated otherwise.

The accuracy of distributional forecasting is assessed using Continuous Ranked Probability Score (CRPS) defined as:

$$\text{CRPS}(\hat{F}, y_o) = \int_{\mathbb{R}} \left(\hat{F}(y) - \mathbb{1}\{y \geq y_o\} \right)^2 dy, \quad (1)$$

where y_o is the observed value, $\hat{F}$ denotes the empirical predictive cumulative distribution function (CDF) determined from the predicted set of quantiles and $\mathbb{1}$ denotes the indicator function.

3. From classical forecasting to learnable information fusion strategies

In the previous section, we introduced a forecasting framework. We now examine how this framework relates to the problem of information fusion — specifically, the fusion of statistical evidence and uncertainty patterns across structurally related but distinct forecasting tasks. To motivate this perspective, we contrast classical forecasting approaches with modern learnable models through the lens of information aggregation and reuse.

Classical statistical models such as ARIMA or ETS generate forecasts by modeling parameters ψ_i for each individual time series i based solely on its historical observations $\mathbf{x}_i$, under a manually selected model structure $\mathcal{M}$ (e.g. ARIMA, ETS), leading to the following Bayesian posterior model (we leave out the constant normalizing terms for brevity):

$$p(\psi_i | \mathbf{x}_i; \mathcal{M}) \propto p(\mathbf{x}_i | \psi_i; \mathcal{M}) p(\psi_i; \mathcal{M}). \quad (2)$$

Here, model parameters ψ_i are estimated independently for each time series, without leveraging information from related series. In

effect, each forecast is produced in isolation, and cross-series structures, e.g. shared temporal dynamics, seasonality profiles, or uncertainty regimes, are ignored. This results in a strictly local inference procedure:

$$p(y_i | \mathbf{x}_i; \mathcal{M}) = \int_{\psi_i} p(y_i | \psi_i; \mathcal{M}) p(\psi_i | \mathbf{x}_i; \mathcal{M}) d\psi_i. \quad (3)$$

In contrast, neural forecasting models trained across collections of time series can learn to fuse statistical signals across series by extracting common patterns and leveraging shared structure. This is particularly valuable in large-scale forecasting settings where many related series (e.g. from different regions, products, or market segments) exhibit similar but non-identical behaviors. From a Bayesian viewpoint, classical models impose fixed priors over time series structure, while data-driven models can learn task-conditioned priors or directly amortize the inference of model parameters.

To formalize this, we introduce a latent variable ϕ that encodes shared statistical knowledge learned from a population of time series $\mathcal{D}_{-i} = \{\mathbf{x}_j\}_{j \neq i}$. This leads to the following hierarchical Bayesian formulation:

$$p(\psi_i, \phi | \mathcal{D}) \propto p(\mathbf{x}_i | \psi_i) p(\psi_i | \phi) p(\mathcal{D}_{-i} | \phi) p(\phi). \quad (4)$$

Here, $p(\psi_i | \phi)$ defines a learned, task-conditional prior over per-series parameters. In practice, this can be instantiated via time-series-specific latent variables such as level and seasonality parameters (see Eqs. (11), (12) in Section 5.1). The term $p(\mathcal{D}_{-i} | \phi)$ captures the global structure shared across the forecasting domain. This global structure can be instantiated by expressive neural backbones (e.g., RNNs or N-BEATS) co-trained across the collection of time series, as detailed in Sections 5.1 and 5.2.

This formulation captures the essence of cross-learning: forecasts for each series are informed not only by their local history but also by statistical structure extracted from related series, yielding the following global inference procedure:

$$p(y_i | \mathbf{x}_i, \mathcal{D}_{-i}) = \int_{\psi_i, \phi} p(y_i | \psi_i, \phi) p(\psi_i, \phi | \mathcal{D}) d\psi_i d\phi. \quad (5)$$

In this sense, ϕ serves as an *information fusion variable*, encoding and propagating inductive bias across tasks. Jointly learning ϕ and per-series parameters ψ_i enables information integration across multiple axes – time series identity, temporal resolution, and predictive uncertainty – yielding a flexible and expressive forecasting paradigm.

4. Any-quantile forecasting framework

The Bayesian formulation above provides a principled foundation for learning to forecast by fusing information across related time series. In particular, it highlights how shared structure, captured via the latent variable ϕ , can be integrated into a forecasting model to support transferability, generalization, and robustness. Building on this foundation, we now introduce a concrete instantiation of this paradigm through our *any-quantile forecasting framework*. This framework implements the fusion mechanism via a shared neural backbone (representing ϕ) and a flexible any-quantile approach shown in Fig. 1, enabling probabilistic forecasts at arbitrary quantile levels using a single model.

4.1. Any-quantile learning

The proposed general methodology enabling any-quantile neural network forecasting is inspired by [38] and is depicted in Fig. 1. The key feature of any-quantile forecasting is that both the neural network and the loss function accept quantile level q as input. Suppose y is observed value, $\hat{y}_q$ is the predicted q th quantile and the neural network is trained using quantile loss, defined as:

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

Then the learning procedure defined based on this framework will possess interesting theoretical properties. In the following analysis we consider for simplicity and without loss of generality that $\mathbf{x}$ is the observed time-series history and y is the unobserved scalar future value that we would like to forecast using a neural forecaster f_θ trained on a N -sample dataset of $(\mathbf{x}, y)$ tuples derived from the unknown joint probability density function $p(\mathbf{x}, y)$. Assuming that at training time quantile probability q is generated uniformly at random in $[0, 1]$ in the mini-batch of size B and the following stochastic gradient descent (SGD) update, based on the quantile loss, is used to update neural forecaster parameters θ (representing both per-series and information fusion parameters, ψ_i and ϕ , respectively), at iteration k :

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \frac{1}{B} \sum_i \rho(y_i, f_\theta(\mathbf{x}_i, q_i)), \quad (7)$$

then the sequence of θ updates linearly converges to the solution of optimization problem on the full training dataset of size N [39]:

$$\theta^* = \arg \min_{\theta \in \Theta} \frac{1}{N} \sum_i \rho(y_i, f_\theta(\mathbf{x}_i, q_i)). \quad (8)$$

By the strong law of large numbers, as N increases without bound, the sum in the last equation converges to the following with probability 1:

$$\mathbb{E}_{\mathbf{x}, y} \mathbb{E}_q \rho(y, f_\theta(\mathbf{x}, q)) = \mathbb{E}_{\mathbf{x}, y} \int_0^1 \rho(y, f_\theta(\mathbf{x}, q)) dq. \quad (9)$$

Recalling that CRPS can be alternatively defined via the inverse CDF F^{-1} of y via quantile loss integral of the form [40]:

$$\text{CRPS}(\hat{F}, y) = 2 \int_0^1 \rho(y, F^{-1}(q)) dq, \quad (10)$$

we establish a correspondence between the mini-batch update in Eq. (7), its expected optimization loss in Eq. (9), and the integral form of CRPS in Eq. (10), thereby linking the model output $f_\theta(\mathbf{x}, q)$ to the inverse of the posterior cumulative distribution function $P_{y|\mathbf{x}}$. That is, for a uniformly sampled quantile $q \sim \mathcal{U}(0, 1)$, the prediction $\hat{y}_q = f_\theta(\mathbf{x}, q)$ corresponds to a sample from the posterior distribution $p(y|\mathbf{x})$.

5. Any-quantile forecasting models

This section describes our main architectural contributions, showing the application of the general any-quantile forecasting methodology outlined in Section 4 to two popular neural forecasting architectures: ESRNN [41] and N-BEATS [36]. ESRNN and N-BEATS represent very different architectural approaches to the forecasting problem. The former is based on recurrent neural network design while the latter has multi-layer perceptron as its core. It is fair to say that these neural network architectures are orthogonal in their way of modeling time series. One of the objectives of our study is to present evidence that the proposed any-quantile learning framework generalizes well over different architectures. In the remainder of this section, we provide architectural design approaches for these two orthogonal neural architectures.

Both proposed models are trained in a cross-learning (or global learning) regime, where a single model is jointly trained on multiple related time series rather than on each series in isolation. By pooling information from diverse but related series, cross-learning enables the model to better generalize and improve forecasting performance across all included series [42,43]. This approach allows the model to capture common structures and dependencies, which is particularly valuable for forecasting electricity demand across multiple countries. In our case, by fusing time-series data from 35 national power systems, the models are exposed to a broad spectrum of uncertainties arising from regional weather patterns, consumption behaviors, and market dynamics. This fusion of diverse sources of uncertainty during training enables the models to learn more robust probabilistic representations, which is crucial for accurate and reliable quantile-based forecasting. In Section 6, we provide empirical evidence that both ESRNN and N-BEATS, when enhanced with the any-quantile learning mechanism and trained in this cross-learning setting, achieve state-of-the-art distributional accuracy in short-term electricity demand forecasting.

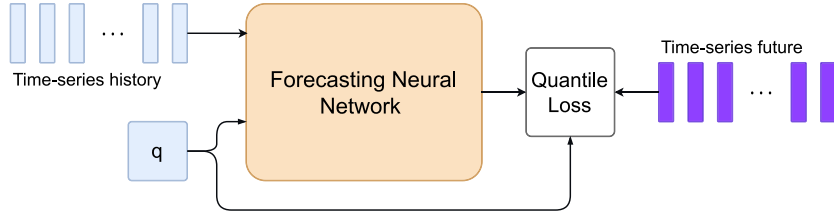


Fig. 1. Any-quantile forecaster training methodology. The network accepts time-series history and a quantile level. The quantile level is generated randomly for each training sample, according to a pre-specified distribution (e.g. uniform or Beta) and is used both as neural network input and as the supervision signal in the quantile loss.

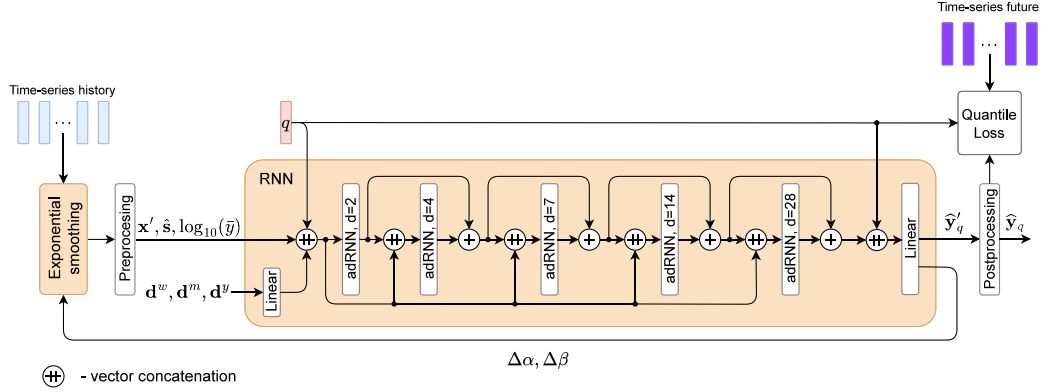


Fig. 2. AQ-ESRNN architecture.

5.1. AQ-ESRNN

The any-quantile enabled ESRNN architecture, which we present in this section and call AQ-ESRNN is derived from ESRNN [41], a hybrid of Exponential Smoothing and a Recurrent Neural Network. Hence, preprocessing, postprocessing and the general architectural approach of AQ-ESRNN and ESRNN are very similar. Therefore, in our current exposition we provide only necessary details and refer interested reader to [41] for a more in-depth elucidation.

Exponential smoothing and pre-processing. In the context of the short-term electricity forecasting, we observe that the hourly time-series exhibit several seasonalities including daily (24 h), weekly (7 days, 168 h), yearly (changing seasons). The high-frequency seasonalities are explicitly treated in AQ-ESRNN by the exponential smoothing and pre-processing blocks shown in Fig. 2 using the $w = 168$ point lag, while the yearly seasonality is dealt with by appending to the input an embedded week number. The exponential smoothing block is based on a version of Holinters multiplicative Exponential Smoothing (ES) model [44]:

$$\begin{aligned} l_{t,\tau} &= \alpha_t \frac{y_\tau}{s_{t,\tau}} + (1 - \alpha_t) l_{t,\tau-1}, \\ s_{t,\tau+w} &= \beta_t \frac{y_\tau}{l_{t,\tau}} + (1 - \beta_t) s_{t,\tau}, \end{aligned} \quad (11)$$

where y_τ is the system load at hour τ , $l_{t,\tau}$ is a level component, $s_{t,\tau}$ is a weekly seasonal component and $\alpha_t, \beta_t \in [0, 1]$ are smoothing coefficients. Note that in contrast to the standard ES, the smoothing coefficients α_t, β_t change with time t . They are calculated from the RNN outputs, $\Delta\alpha_t$ and $\Delta\beta_t$:

$$\begin{aligned} \alpha_{t+1} &= \sigma(\alpha_0 + \Delta\alpha_t), \\ \beta_{t+1} &= \sigma(\beta_0 + \Delta\beta_t). \end{aligned} \quad (12)$$

Here α_0, β_0 are initial values of the smoothing coefficients and σ is a sigmoid function, which maintains the coefficients within $(0, 1)$ range.

Note in Fig. 2 that the RNN produces not only the forecasts of the target quantile, $\hat{y}_q$, but also corrections to the ES parameters, $\Delta\alpha_t$ and $\Delta\beta_t$. These corrections are used to dynamically update the seasonal

component $s_{t,\tau}$, which is then applied during the preprocessing step – see (13). In this way, the model leverages the RNN to dynamically adjust the ES parameters, effectively optimizing the preprocessing step. This mechanism is designed to enhance the performance of the RNN in quantile forecasting and, consequently, to improve the overall forecasting accuracy.

Recurrent NN backbone is a deep residual RNN, which we call RNN for simplicity. RNN has several inputs, including the randomly generated quantile level q and a normalized, deseasonalized vector of the past $w = 168$ values:

$$x_\tau = \log \frac{y_\tau}{\bar{y}_t \hat{s}_{t,\tau}}, \quad (13)$$

where $\tau \in \Delta_t^{in}$, Δ_t^{in} denotes the input rolling window, which is adjacent to the output window Δ_t^{out} of length $H = 48$ covering the forecast horizon, $\bar{y}_t$ is the average value in Δ_t^{in} and $\hat{s}_{t,\tau}$ is the seasonal component produced by ES.

The input vector concatenates q and $\mathbf{x}'_t = [x_\tau]_{\tau \in \Delta_t^{in}}$ with: (i) date features, $\mathbf{d}_t^w \in \{0, 1\}^7$, $\mathbf{d}_t^m \in \{0, 1\}^{31}$ and $\mathbf{d}_t^y \in \{0, 1\}^{52}$, i.e. binary one-hot vectors encoding day of the week, day of the month and week of the year, respectively; (ii) seasonality components, $\hat{s}_t$, i.e. a vector of 48 seasonal components predicted by ES for the output period t reduced by 1 and (iii) logarithm of the normalizer, $\log_{10}(\bar{y}_t)$.

The input is fed to the deep dilated RNN architecture consisting of 5 recurrent layers as shown in Fig. 2. RNN employs attentive dilated RNN cells, adRNNCells, introduced in [41]. They possess a receptive field whose size is determined by a dilation parameter ($d \geq 1$). Moreover, they incorporate an attention mechanism that dynamically adjusts the weighting of input information, making them particularly effective at capturing complex dynamics and seasonalities. Larger dilations are assigned to top layers, i.e. $d = 2, 4, 7, 14, 28$. The dilations are in day units, as we step in time by 24 h. Residual connections flowing around each adRNNCell are added to avoid vanishing gradient problem. Finally, an important feature of the architecture is that the input containing a reference to the quantile probability q is repeatedly appended to the input of each layer, which improves the distributional forecasting accuracy according to our ablation studies in Section 6.7.

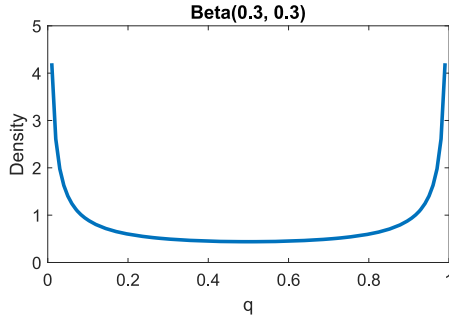


Fig. 3. Beta(0.3, 0.3) distribution used for the training of AQ-ESRNN. It oversamples the quickly changing, and important, edge areas.

Comparing the ESRNN and AQ-ESRNN architectures, we note that although the output size of AQ-ESRNN is significantly smaller (it forecasts a single quantile rather than multiple quantiles, e.g., five in the case of ESRNN), it is nonetheless deeper, employing five recurrent cells compared to three in ESRNN. This reflects the fact that forecasting any arbitrary quantile on demand is a more challenging task.

Postprocessing. The postprocessing component converts the RNN output, represented as a vector of $H = 48$ normalized and deseasonalized q th quantiles, $\hat{y}'_{q,t}$, into actual quantiles, $\hat{y}_{q,t}$, using inverted Eq. (13):

$$\hat{y}_{q,t} = \exp(\hat{y}'_{q,t}) \bar{y}_t \hat{f}_{t,\tau}, \quad \tau \in \Delta_t^{out}. \quad (14)$$

We also found empirically that postprocessing the RNN forecast via quantile sorting noticeably improves distributional forecasting accuracy. For every horizon length (1 to 48) we sort the forecasted values for all quantiles and then match them with the sorted quantile probabilities. This prevents the quantile crossing and greatly improves the forecast accuracy (please refer to ablation studies for details).

Training Methodology. We use standard quantile loss function (6), operating on normalized values, i.e. $y^n = y/\bar{y}$ and $\hat{y}_q^n = \hat{y}_q/\bar{y}$. A unique quantile probability q is sampled randomly for each time-series in the batch. It is worth mentioning that we found that AQ-ESRNN training benefits from non-uniform sampling of q with emphasis on the tails. We attribute this to the fact that ESRNN uses saturating activation functions, which lead to diminishing gradients at the tail of the distribution. Thus it is more challenging to fit well to very small (close to 0) and very large (close to 1) quantiles, because the quantile function is likely to change quickly in those regions. Additionally, these edge quantile levels are often very important for planning purposes. We conjecture that this can be compensated for by denser sampling of quantile probabilities in the tails. In particular, we use symmetric Beta distribution with both shape parameters less than 1, e.g., Beta(0.3, 0.3) (see Fig. 3).

5.2. AQ-NBEATS

Our exposition of the original N-BEATS architecture closely follows [37]. Each block of N-BEATS is a residual architecture consisting of the sequence of fully connected blocks with forecast/backcast forks at the end. The architecture runs a residual recursion over the entire input window and sums block outputs to make its final forecast. We assume that there are R residual blocks each having L hidden layers. If we refer to $\mathbf{x} \in \mathbb{R}^W$ as the input of the architecture of length W , use residual block and layer superscripts (r and ℓ , respectively) and denote the fully connected layer with weights $\mathbf{W}^{r,\ell}$ and biases $\mathbf{b}^{r,\ell}$ as $\text{FC}_{r,\ell}(\mathbf{h}^{r,\ell-1}) \equiv \text{ReLU}(\mathbf{W}^{r,\ell} \mathbf{h}^{r,\ell-1} + \mathbf{b}^{r,\ell})$, the operation of N-BEATS is described as follows:

$$\begin{aligned} \mathbf{x}^r &= \mathbf{x}^{r-1} - \hat{\mathbf{x}}^{r-1}, \\ \mathbf{h}^{r,1} &= \text{FC}_{r,1}(\mathbf{x}^r), \dots, \mathbf{h}^{r,L} = \text{FC}_{r,L}(\mathbf{h}^{r,L-1}), \\ \hat{\mathbf{x}}^r &= \mathbf{B}^r \mathbf{h}^{r,L}, \hat{\mathbf{y}}^r = \mathbf{F}^r \mathbf{h}^{r,L}. \end{aligned} \quad (15)$$

Here we have $\hat{\mathbf{x}}^0 \equiv \mathbf{0}$, $\mathbf{x}^0 \equiv \mathbf{x}$ and projection matrices have dimensions $\mathbf{B}^r \in \mathbb{R}^{W \times d_h}$, $\mathbf{F}^r \in \mathbb{R}^{H \times d_h}$. Internal hidden dimension of each block is d_h . The final forecast is the sum of forecasts of all the residual blocks, $\hat{\mathbf{y}} = \sum_r \hat{\mathbf{y}}^r$. Given the structure of the network, there are three principal ways, depicted in Fig. 4 and described in detail below, that can be used to feed quantile q to N-BEATS.

Input quantile concatenation scheme is the easiest (does not require architectural modifications) and the most generic approach that is applicable to N-BEATS as well as any other forecasting architecture. In this scheme, called AQ-NBEATS-CAT and depicted in Fig. 4(a), quantile level is simply appended as one of the elements (e.g. the last element) in the input time-series history. The caveat here is that in the univariate forecasting scenario the original input has dimensions $B \times W$ (batch size times the history length). In order to enable quantile computation for multiple quantiles Q for each time-series, the quantile-appended input will now have the form of 3D tensor of the size $B \times Q \times (W + 1)$, which multiplies the total computational cost of predicting Q quantiles at inference time by a factor of Q . Additionally, quantile level is restricted to the range $[0, 1]$. Hence, concatenating it with inputs of large magnitude can create dynamic range issues and make the network completely ignore the q input. Therefore, additional input normalization measures may have to be necessary to ensure effective quantile operation.

Block quantile conditioning scheme depicted in Fig. 4(b) is more specific to N-BEATS and implies architectural modification. In this scheme, which we call AQ-NBEATS-FiLM, we propose to use the FiLM conditioning method [45] to modulate the computation of each N-BEATS block with the representation derived from the quantile probability. In terms of inference path, the first and the last equation of AQ-NBEATS-FiLM block replicate original N-BEATS (15), whereas the middle part is modified as follows:

$$\begin{aligned} \mathbf{h}^{r,1} &= \text{FC}_{r,1}(\mathbf{x}^r), \\ \mathbf{c}_\alpha^r &= \text{LINEAR}_{r,\alpha}(\mathbf{q}), \quad \mathbf{c}_\gamma^r = \text{LINEAR}_{r,\gamma}(\mathbf{q}), \\ \mathbf{h}^{r,2} &= \mathbf{c}_\alpha^r + (1 + \mathbf{c}_\gamma^r) \mathbf{h}^{r,1}, \\ &\dots \\ \mathbf{h}^{r,L} &= \text{FC}_{r,L}(\mathbf{h}^{r,L-1}). \end{aligned} \quad (16)$$

FiLM induces both additive shift and multiplicative modulation of the internal time-series representation based on the target quantile probability. Since both multiplicative and additive components are included, this scheme is less sensitive to the issues that may arise from the lack of normalization, the problem we pointed out in relation with the AQ-NBEATS-CAT variant. Additionally, we hypothesize that the inclusion of quantile conditioning in each block should lead to better overall modeling of quantiles, since the architecture is effectively “reminded” of the quantile forecasting task in every block, whereas in AQ-NBEATS-CAT, the quantile hint is provided only once at the input and by the time the representation reaches the last block of a deep network it can effectively be washed away.

Output quantile conditioning scheme called AQ-NBEATS-OUT is depicted in Fig. 4(c). Here the bulk of the processing is done in a standard forecasting architecture, while quantile conditioning is done in the special output quantile conditioning block. This is a more computationally effective variant of AQ-NBEATS-FiLM. In AQ-NBEATS-FiLM, there is a computational cost for constantly “reminding” the architecture about the quantile it has to produce. Since quantile conditioning happens in every block, the architecture persistently works with $B \times Q \times d_h$ tensors (d_h is the dimensionality of hidden layers), multiplying the total computational cost of predicting Q quantiles at inference time by a factor of Q . We hypothesize that the bulk of the work of representing a time-series can be done without quantile conditioning. Therefore, if we apply quantile conditioning only in the last block, as depicted in Fig. 4(c), we should be able to obtain comparable accuracy results while significantly saving on computation. For example, with a typical configuration of N-BEATS depth of 10–30 blocks, the computational

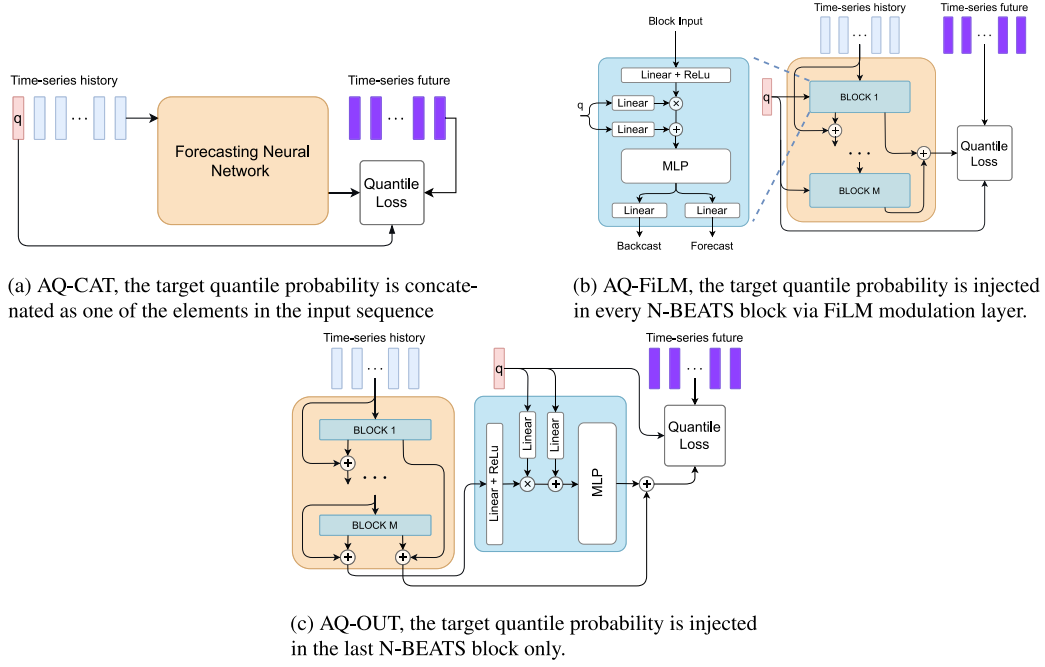


Fig. 4. Variants of implementing the any-quantile conditioning in N-BEATS.

cost of AQ-NBEATS-OUT will be 10–30 times smaller than that of AQ-NBEATS-FiLM.

Training Methodology. To train AQ-NBEATS, we use standard quantile loss function (6). A unique quantile probability q is sampled for each time-series in the batch, uniformly at random. The details of training methodology along with hyperparameter settings are provided in Section 6.4.

6. Empirical results

In this section, we conduct a comprehensive assessment of our proposed AQ-models in the context of short-term electricity demand forecasting across 35 European countries. The evaluation process unfolds as follows: we provide insights into the nature of the data, elucidate the methodologies involved in training and evaluation and describe models used as baselines. Subsequently, we present the results, delve into an ablation studies and conclude by engaging in a thorough discussion regarding the distinctive properties of our models.

6.1. Datasets

The real-world dataset utilized in this study is sourced from the ENTSO-E repository (www.entsoe.eu/data/power-stats). This dataset encompasses hourly electricity demand data for 35 European countries and we have made this dataset publicly available along with the model code in our GitHub repository (To be released if the paper is accepted). The dataset contains a rich collection of time-series, each possessing distinct characteristics such as level, trend and variance varying over time, seasonalities (annual, weekly and daily) and random fluctuations of varying intensity. The inclusion of time-series with such diverse properties ensures a robust evaluation of the model's performance.

Fig. 5 provides a snapshot of the diverse energy consumption profiles across European countries. It illustrates the average demand and daily dispersion of hourly demands over the three-year period from 2016 to 2018. The average demand exhibits notable variability, ranging from 383 MWh for Montenegro (ME) to 58,941 MWh for Germany (DE). Similarly, the daily standard deviation of hourly demands varies from 56 MWh (Iceland, IS) to 7687 MWh (DE).

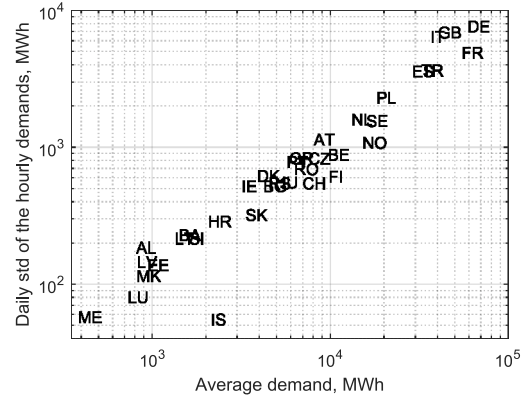


Fig. 5. Average electricity demand and daily dispersion demand in 35 European countries.

The variability in demand significantly influences the width of the probabilistic forecast distribution. We quantify this variability as the ratio of the standard deviation to average demand value: $v = 100s/\bar{z}$. We consider it in three contexts: as daily demand variations (v_d), where $\bar{z}$ and s denote the daily mean and standard deviation, respectively; as weekly variations (v_w), where $\bar{z}$ and s represent the weekly mean and standard deviation of daily means; and as yearly variations (v_y), where $\bar{z}$ and s express the yearly mean and standard deviation of weekly means. Fig. 6 shows the variation coefficients for 35 European countries. Notably, Iceland exhibits the least demand variation. The most pronounced daily variations ($v_d > 17\%$) are observed in Albania, Italy, Latvia and Great Britain. For yearly variations, Norway, France, Sweden and Macedonia demonstrate the highest variability ($v_y > 18\%$). Weekly variations are generally milder than daily and yearly ones, with $v_w < 13\%$. Italy, Germany, Austria and Poland show the strongest weekly variations. It is crucial to highlight that the pattern of electricity demand variation evolves over time, presenting additional challenge for forecasting models.

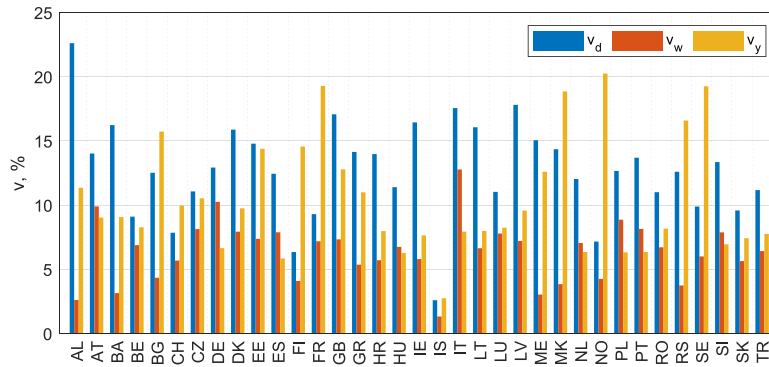


Fig. 6. Diversity of electricity demand in 35 European countries.

6.2. Training and evaluation setup

The dataset is split into train, validation and test subsets with the following boundaries: train dates 2006-01-01 to 2016-12-31, validation dates 2017-01-01 to 2017-12-31, test dates 2018-01-01 to 2018-12-31. In the training set, NaN (missing) values are forward-filled by replacing each missing value with the value from 168 h earlier (i.e., one full week) to enable their use during training. This is arguably the most suitable method for handling missing values when the time series exhibits strong high-frequency seasonality (weekly, with a 168-h cycle), along with an additional low-frequency seasonality (yearly). In the test set, NaN values (which occur during the final month of 2018 for Estonia and Italy, and the last two months of 2018 for Latvia) are excluded from metric calculations to avoid any information leakage.

We use the train and validation subsets to tune hyperparameters. Once the hyperparameters are determined, the model with best hyperparameter settings, re-trained on the train and validation sets, is used to report results on the test set. The prediction horizon is 48, or two days. A prediction horizon of this length is imposed by the requirements of power system operation. The system operator typically prepares forecasts for the next day well in advance, for example 12 h in advance according to [46].

For the final evaluation, we employ the ensemble of eight models, each sharing the same structure but initialized randomly and exposed to distinct, randomized sequences of input data (for a fair comparison, all baseline neural models are likewise evaluated using ensembles of eight models). The final forecasts are generated by taking the medians of all ensemble forecasts.

ESRNN model training and inference is performed on a PC with Intel i9-10980XE CPU and 256 GB of RAM. We train the ensemble (8 models) concurrently. In this setup the training of full models with 5 layers takes around 2–3 days. N-BEATS model training and inference are performed on the Dell Poweredge C4130 with 4xNVIDIA P100 GPU and 2 x Intel Xeon E5-2660 v3 @ 2.6 GHz processors. The training time in this hardware setup is 8 h and inference time is 10 min for 8-model N-BEATS AQ-OUT ensemble.

Statistical and machine learning baseline models are trained on eight-core CPU (AMD Ryzen 7 1700, 3.0 GHz, 32 GB RAM) resulting in the following compute times per model: ARIMA: 180 h; ES: 50 h; Theta: 30 h; QR: 58 h; NGBoost: 20 h; MLP: 6 min; DeepAR: 2 h and 40 min; Transformer: 3 h and 10 min; TFT: 2 h and 50 min; WaveNet: 7 h.

6.3. AQ-ESRNN training and hyperparameter settings

All AQ-ESRNN hyperparameters are adjusted by minimizing CRPS on the validation set. The objective function used to train the network is quantile loss (6). The model is implemented in PyTorch [47] and is trained for 6 epochs corresponding to around 1500 batch updates using Adam optimizer [48]. We use a schedule of increasing batch sizes.

Table 1

Settings of AQ-ESRNN hyperparameters and the hyperparameter search grid.

Hyperparameter	Value	Grid
Batch size	2, changed to 5 in ep. 4	changed to 5 in ep. 3, 4
Epochs	6	4, ..., 9
Learning rates	0.003, /3 in ep. 5, /10 in ep. 6	also /30 in ep. 7
LR multiplier	30	[10, 30]
BetaShape	0.3	[0.1, 0.2, 0.3, 0.5, 0.65, 0.8, 1]
Dilations	[[2],[4],[7],[14],[28]]	See Section 6.3
training steps	50	[20, 30, 50]
c-state size	150	[100, 150, 200]
h-state size	70	[50, 70, 100]
α_0	3.5	–
β_0	0.3	–

Initially, we use batch size of 2 corresponding to 1750 updates per epoch and as batch size is increased to 5, the number of batch updates decreases to approximately 1450 per epoch. In addition to batch size scheduling, we use learning rate scheduling. On top of this, model contains global learnable parameters and time-series specific learnable parameters. For the latter, there is LR multiplier boosting the learning rate for per-series parameters. When experimenting with models having fewer layers, we observed earlier signs of overfitting. Therefore, for 3 and 4-layer models, we report results from epoch 5, while for 5-layer models, we report results from epoch 6.

Table 1 summarizes optimal hyperparameter settings of the AQ-ESRNN model, alongside the ranges explored during hyperparameter tuning process. A few additional remarks are in order to explain the fields of the table. *BetaShape* is a parameter utilized for a symmetrical Beta distribution, which is employed in generating the quantiles during training. *Dilations* describe the architecture of AQ-ESRNN. Each internal square bracket lists the dilations of layers in a particular block. For instance, [[2], [4], [7], [14], [28]] describes a system composed of 5 blocks, each with a single layer and a progressively increasing delay of 2, 4, 7, 14 and 28 time steps. In experiments with 3-layer architecture, dilations are set as [[2], [4], [7]]. For 4-layer architecture, we experimented with dilations like [[2], [4], [7], [14]], [[2], [4], [7], [7]] and [[2], [4], [7], [4]]. Similarly, for 5-layer architectures, we tested dilations such as [[2], [4], [7], [14], [28]], [[2], [4], [7], [14], [14]], [[2], [4], [7], [14], [7]] and [[2], [4], [7], [10], [14]]. *Training steps* – when a batch is formed from randomly chosen series, the RNN time steps initiate from a randomly selected step and persist for this number of steps. *C-state size* and *h-state size* – the cells employed are derived from LSTMs, but differ in several aspects, including c-state and h-state having different sizes. The c-state size must be larger than h-state size, because the cell output size equals c-state size minus h-state size.

6.4. AQ-NBEATS training and hyperparameter settings

AQ-NBEATS model is evaluated using the hyperparameter settings presented in Table 2. Similarly to AQ-ESRNN, the hyperparameters

Table 2
Settings of AQ-NBEATS hyperparameters and the hyperparameter search grid.

Hyperparameter	Value	Grid
Epochs	15	[10, 15, 20]
Loss	Quantile, (17)	[(17), (6)]
Input Normalization	No	[Yes, No]
Width (d_h)	1024	[256, 512, 1024]
Blocks (R)	30	[5, 10, 20, 30]
Layers (L)	3	[2, 3, 4]
Sharing	False	[True, False]
Lookback period (w , hours)	168	[24, 168, 336]
Batch size	1024	[256, 512, 1024]
Optimizer	Adam	Adam
Learning rate	0.0005	[0.001, 0.0005]

are adjusted by minimizing CRPS on the validation set. The objective function used to train the network is normalized quantile loss:

$$\rho_N(y, \hat{y}_q) = \begin{cases} (y - \hat{y}_q)q/y, & \text{if } y \geq \hat{y}_q, \\ (y - \hat{y}_q)(q-1)/y, & \text{if } y < \hat{y}_q. \end{cases} \quad (17)$$

The model is implemented in PyTorch [47] using pytorch lightning framework [49] trained for 15 epochs using the Adam optimizer [48] with default settings using inverse square root learning rate annealing with maximum learning rate 0.0005 and warm-up period of 400 batches. Batch has size 1024 and the model takes the history of 168 points (24 h times 7 days) and predicts 48 points (two times 24 h) ahead in one shot.

6.5. Baseline models

We compare our AQ-models against a number of probabilistic forecasting baselines, encompassing both statistical and machine learning models, primarily neural network-based approaches. First, we list the baselines and in the ensuing paragraphs we explain important implementation details.

- **Naive**, a seasonal naive method: the forecasted demand profile for day t is the same as the profile for day $t-7$.
- **ARIMA**, autoregressive integrated moving average model [50].
- **ES**, exponential smoothing model [50].
- **Theta**, dynamic optimized Theta model, DOTM v3 from [51].
- **QR**, quantile linear regression model [52].
- **NGBoost**, natural gradient boosting model [53].
- **MLP**, single-hidden-layer perceptron with sigmoid nonlinearities [54].
- **DeepAR**, autoregressive RNN model for probabilistic forecasting [55].
- **Transformer**, vanilla transformer [56].
- **WaveNet**, deep autoregressive model combining causal filters with dilated convolutions [57].
- **TFT**, temporal fusion transformer [58].

Naive, ARIMA, ES and Theta models generate point forecasts. To deal with triple seasonality in electricity demand time-series, we decompose these series into 24 distinct series, each representing a particular hour of the day. For each of these 24 series, a separate statistical model is utilized to forecast electricity demand, with a horizon of two. Consequently, all models produce a total of 48 forecasts for the subsequent two days. For Naive, ARIMA and ES, we assume that the point forecast represents mean of normal distribution and the standard deviation is estimated using the standard deviation of residuals [59]. To obtain quantiles, the models are first fitted using their respective R functions in the forecast package: `snaive()`, `auto.arima()` and `ets()`. Quantiles are then derived by setting the `level` parameter in the `forecast()` function to specific value defining PI. In contrast,

Table 3
Key performance metrics. Lower score is better. Best result is shown in bold.

	CRPS	N-CRPS	MAPE
Naive	502.62	3.95	5.08
ARIMA	353.49	2.83	3.74
ES	325.80	2.64	3.48
Theta	329.99	2.69	3.56
QR	357.38	2.90	3.92
NGBoost	329.21	2.72	3.75
MLP	415.67	3.28	4.49
DeepAR	378.46	2.97	3.95
WaveNet	293.38	2.52	3.39
Transformer	380.41	2.83	3.72
TFT	325.80	2.92	3.96
AQ-ESRNN	195.94	1.72	2.32
AQ-NBEATS	211.22	1.84	2.47

the Theta method estimates quantiles through bootstrapping, which involves simulating a sample of possible values of forecasts from the estimated model [60]. This technique is implemented in the `dotm()` function in the `forecTheta` package.

The QR implementation utilizes QuantileRegressor from the `sklearn.linear_model` Python library [61], while NGBoost forecasts are generated using NGBRegressor from the `ngboost` package. Both these models incorporate comprehensive feature sets consisting of calendar variables (hour of day, day of week, quarter, month, year, day of year, day of month, and week of year) and historical demand patterns. The historical features capture lagged demand values from $t-2$ to $t-21$, representing 20 consecutive days of demand history preceding the forecast period. For each quantile level, a separate QR model is trained using the pinball loss function. In contrast, NGBoost generates quantile forecasts by estimating the parameters of a parametric probability distribution, specifically, the normal distribution in our case. Once the distribution parameters are predicted, the quantiles are obtained analytically from the corresponding inverse cumulative distribution function. NGBoost is trained by minimizing the negative log-likelihood of the target variable under the predicted distribution, using natural gradient descent.

Neural models employed in this study are sourced from GluonTS, a Python package designed for probabilistic time-series modeling, with a primary focus on deep learning models [33]. Specifically, the MLP, DeepAR and Transformer models are utilized to generate probabilistic forecasts through a parametric approach. These models have a projection layer that produces distribution parameters. Model fitting process involves the negative log-likelihood of the chosen distribution as the loss function. We use Student's t-distribution, which is the default setting. Once the model is fitted, quantiles can be derived from the learned parametric distribution. In contrast, WaveNet models the auto-regressive forecast probability distribution by outputting a softmax-induced categorical distribution over a fixed grid of quantiles. Finally, TFT simultaneously predicts multiple quantiles at each time step alongside a point forecast. Quantile forecasts are derived by applying a linear transformation on top of the TFT decoder embedding. During training, TFT learns by jointly minimizing the quantile loss aggregated across all quantile outputs. Arbitrary quantiles are derived from the fixed quantile grid via interpolation.

The hyperparameter settings of the baseline models are summarized in Tables A.6–A.9 in the Appendix.

6.6. Key quantitative results

Our key results are presented in Table 3. They include CRPS (1), normalized CRPS (N-CRPS) and MAPE for assessing point forecasts derived from distributional forecasts. Normalized version of CRPS is defined as follows:

$$\text{N-CRPS}(f_{\theta}, \mathbf{y}, \mathbf{x}) = 100 \frac{1}{C} \sum_c \frac{1}{HN_c Q} \sum_{i,h,q} \rho(y_{i,h,c}, f_{\theta}(\mathbf{x}_{i,c}, q)) / \bar{y}_c. \quad (18)$$

Naive	0	0	1	1	0	0	0	0	1	0	2	0	0
ARIMA	35	0	1	4	13	6	27	9	2	7	7	0	0
ES	34	30	0	19	26	10	32	23	4	15	11	0	0
Theta	34	27	10	0	27	6	33	23	4	13	7	0	0
QR	34	8	3	4	0	6	27	11	3	6	7	0	0
NGBoost	35	24	21	23	28	0	32	26	13	25	14	0	1
MLP	35	2	1	2	3	0	0	4	1	2	5	0	0
DeepAR	35	13	4	2	16	4	31	0	2	5	6	0	0
WaveNet	34	31	28	30	30	13	33	33	0	30	20	0	0
Transformer	34	20	7	10	23	8	32	24	2	0	8	0	0
TFT	33	25	21	22	26	13	29	27	13	23	0	0	0
AQ-ESRNN	35	35	35	35	35	35	35	35	34	35	35	0	24
AQ-NBEATS	35	35	35	35	35	34	35	35	35	35	35	5	0

Fig. 7. Results of the Diebolt-Mariano test based on the N-CRPS metric ($\alpha = 0.05$). Individual cells count the number of countries the model represented on the y-axis is statistically more accurate than the model represented on the x-axis.

Here $C = 35$ is the number of countries, $H = 48$ is the number of horizons, $Q = 100$ is the number of random test quantiles and N_c is the number of test samples per country. The second sum runs over samples in the country time series, horizons and quantiles. $\bar{y}_c$ is the mean of a given country time-series. Normalized CRPS unifies results for different countries by normalizing over mean demand level $\bar{y}_c$. Therefore, all countries can be compared with each other and their impact in the average N-CRPS value shown in Table 3 is similar.

We see that the any-quantile forecasting task is challenging for classical time-series models as well as for the out-of-the-box machine learning models such as NGBoost and NNs from GluonTS. Among statistical models, ES emerges as top performer, while WaveNet leads the neural baselines. WaveNet slightly outperforms ES. Our novel any-quantile architectures, AQ-ESRNN and AQ-NBEATS, set new benchmarks in distributional forecasting, achieving the lowest CRPS. Additionally, when comparing mean absolute percentage errors (MAPE) as the performance measure for point forecasting, where the 0.5-quantile (median) is treated as a point forecast, the AQ-models surpass all baselines. Notably, AQ-ESRNN slightly outperforms AQ-NBEATS across both CRPS and MAPE metrics.

To assess the statistical significance of differences between the probabilistic forecasts generated by each pair of models, we conduct a Diebold–Mariano test [62] based on the N-CRPS metric computed for individual countries. The results, evaluated at a 0.05 significance level, are shown in Fig. 7, which counts the number of countries the model represented on the y-axis is statistically more accurate than the model represented on the x-axis. It is noteworthy that both our AQ-models demonstrate superior accuracy over baseline models for nearly all countries. Furthermore, AQ-ESRNN exhibits higher accuracy than AQ-NBEATS in 24 countries, while AQ-NBEATS outperforms AQ-ESRNN in 5 countries.

Fig. 8 presents the evaluation of PIs of varying sizes generated by the forecasting models. Bars indicate the proportion of observed values that fall within, below, or above the boundaries of the PIs predicted by the models, while the red dashed lines signify the theoretical target (for instance, for a 50%-PI, the targets are 50% within, 25% below and 25% above the PIs). In the context of In-PI charts, bars surpassing the target level signify PIs that are excessively broad, implying an overestimation of the uncertainty range. Conversely, bars falling short of target suggest PIs that are unduly narrow, indicating an underestimation of uncertainty. The pattern of too narrow PIs is particularly evident in the cases of the DeepAR and Transformer models, whereas

an inclination towards too broad PIs is primarily observed for 50%-PI in the statistical models, Naive, WaveNet and TFT. When examining the Below-PI and Above-PI charts, the most striking observation is that for DeepAR and Transformer the upper PI bounds are notably too low, with the bars in the Above-PI charts significantly above the target values. Conversely, WaveNet exhibits an opposite trend, with excessively high upper PI bounds. For broader PIs, specifically those at 98% and 99.8%, predicted PIs are often too tight, indicating challenges in forecasting extreme quantiles. In these scenarios, the TFT model exhibits superior performance. Excluding these extreme quantiles, the performance of our any-quantile models stands out, achieving PIs that closely approximate the targets.

To provide for a more comprehensive assessment of probabilistic forecasts, we additionally examine relative frequency (RF), also referred to as calibration or coverage [63]. $RF(q)$ quantifies the proportion of observed values y_i falling below or equal to the predicted q -quantile $\hat{y}_{q,i}$ and is defined as follows:

$$RF(q) = \frac{1}{M} \sum_{i=1}^M \mathbb{1}\{y_i \leq \hat{y}_{q,i}\}. \quad (19)$$

For probabilistic forecasts to be precise and informative, they should exhibit an RF value close to the nominal probability level, with a reasonably small deviation from the realized values [64]. In essence, probabilistic forecasts are expected to capture uncertainty without significantly deviating from the actual series values. Ideally, predicted q -quantiles should surpass the realized values in $100 \cdot q\%$ of cases, ensuring an RF of q . To evaluate the average deviation of $RF(q)$ from the desired q across all assumed quantile probabilities $q \in \Pi = \{0.001, 0.01, 0.02, \dots, 0.99, 0.999\}$, we introduce the mean absolute RF error:

$$MARFE = \frac{1}{|\Pi|} \sum_{q \in \Pi} |RF(q) - q|. \quad (20)$$

Fig. 9 illustrates the RF charts, where the desired RF values are depicted with dashed lines. Each dot represents RF calculated for a specific country and probability $q \in \Pi$, while the red line signifies the mean RF value across all countries. From this figure, it is apparent that DeepAR, WaveNet, Transformer and TFT exhibit the most distorted RF distributions, displaying deviations from the desired probabilities in various directions. MARFE, corresponding to the deviation of the red line from the dashed line, exceeds 0.04 for these models. In contrast, Naive and statistical models demonstrate less scattered results, with a slightly s-shaped curve. The lowest MARFE is observed for QR (0.0104), followed by AQ-ESRNN (0.0157), MLP (0.0168), NGBoost (0.0174), Theta (0.0187) and AQ-NBEATS (0.0211). Notably, our any-quantile models accurately express the target probabilities, with RF curves closely aligned with the desired values and the average line almost coinciding with the dashed line.

Fig. 10 concludes our empirical study by providing the examples of probabilistic forecasts generated by our proposed AQ-models for Poland, Great Britain, France and Germany.

6.7. AQ-ESRNN ablation studies

Ablation studies of AQ-ESRNN are reported in Table 4, with the full model showcased in the first row. Initially, we explore the impact of reducing the number of layers (cells). It is evident that a larger number of layers enhances accuracy, albeit at the expense of increased computation time. Next, AQ-ESRNN-noQuantInMidLayers removes the quantile from the middle layers, while AQ-ESRNN-noInputInMidLayers eliminates all inputs from the middle layers. These architectural variants are depicted in Fig. 11a and b, respectively. Both of these simplified variants yield inferior results, demonstrating the value of “reminding” the target probability to the neural network at each block. Furthermore, AQ-ESRNN-dRNN variant represents the full model (5 layers) but employs simpler dRNN cells without the attention mechanism,

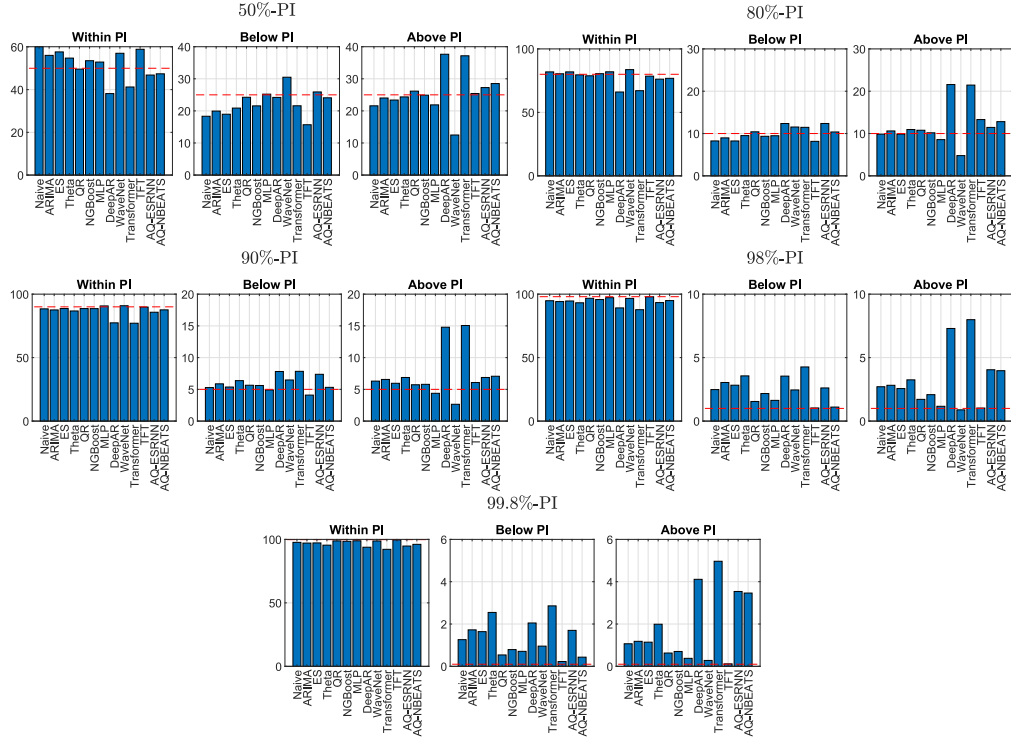


Fig. 8. Percentage of forecasts in, below and above the PI of different size. Red dashed line indicates the target value.

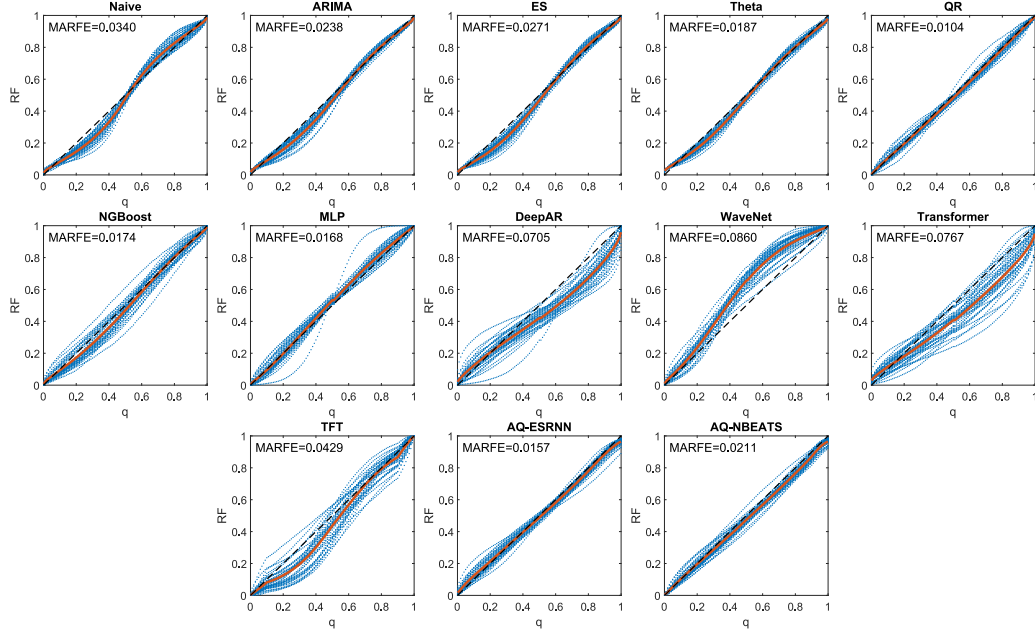


Fig. 9. Relative frequency.

as described in [41]. AQ-ESRNN-LSTM uses the standard LSTM cell. Simplifying the cell type leads to a deterioration in results. Lastly, AQ-ESRNN-unsorted is the full AQ-ESRNN model that does not sort forecast values. Sorting emerges as a crucial element of this model's CRPS performance. Without sorting, CRPS increases significantly, by approximately 34%, compared to the full variant.

6.8. AQ-NBEATS ablation studies

Table 5 studies the influence of the following factors in AQ-NBEATS architecture. First, AQ-NBEATS can work with both normalized and non-normalized inputs. Therefore, we explore the case of normalized input, in which case the input time-series is divided by the absolute

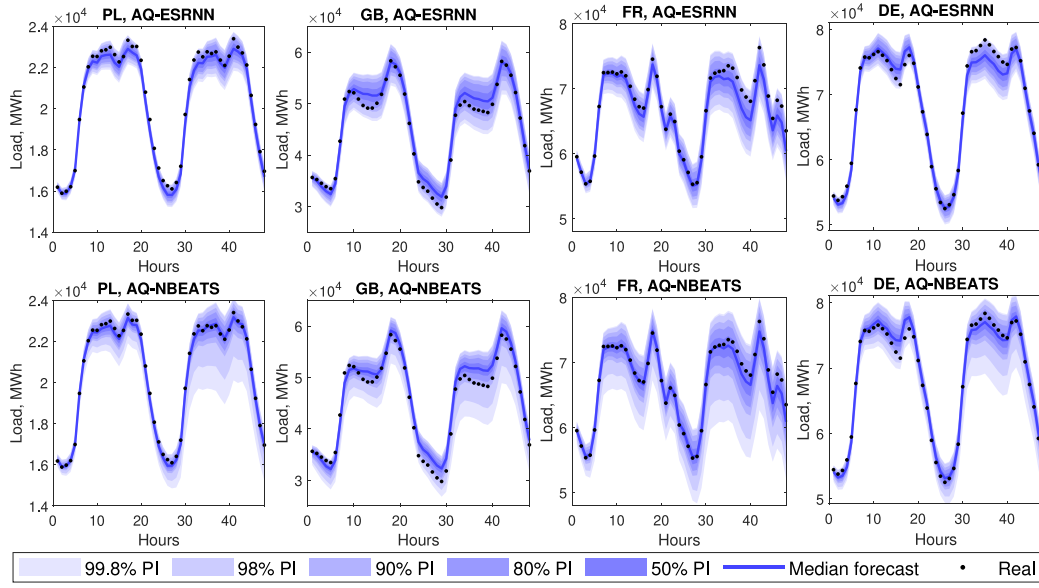


Fig. 10. Examples of quantile forecasts for 30 and 31 January 2018.

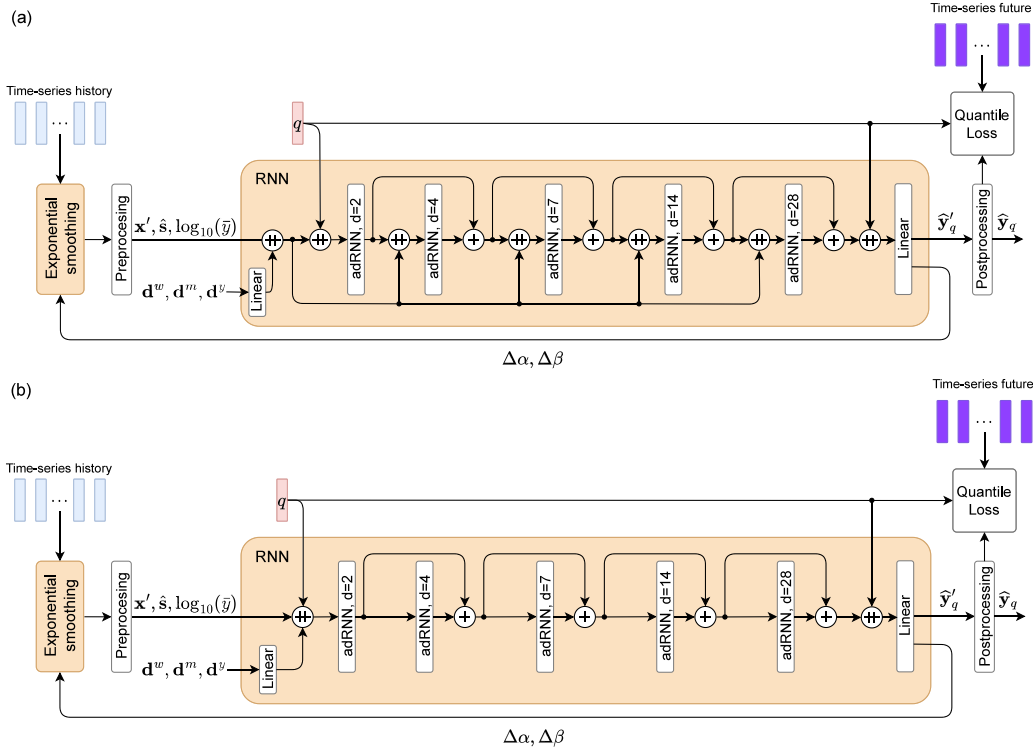


Fig. 11. Reduced versions of AQ-ESRNN: (a) AQ-ESRNN-noQuantInMidLayers, (b) AQ-ESRNN-noInputInMidLayers.

maximum value of the input window (Max-Norm column checkmark) and unnormalized input (Max-Norm column no checkmark). Second, learning supervision can be done either with the MQ loss defined in Eq. (6) (no checkmark in MQN column) or with its normalized MQN variant defined in Eq. (17) (checkmark in MQN column). Finally, we explore the following AQ-NBEATS variants discussed in Section 5.2 and depicted in Fig. 4: (i) AQ-NBEATS-CAT, the quantile level is appended to the input sequence as additional token, (ii) AQ-NBEATS-FiLM, the quantile is inserted in every block through FiLM mechanism, (iii) AQ-NBEATS-OUT, a separate quantile block is appended at the end of the standard N-BEATS.

Looking at Table 5 the following conclusions can be made. First, AQ-NBEATS-FiLM is the most accurate variant, which implies that providing quantile level as input to every block of the architecture is beneficial from the accuracy standpoint. Second, AQ-NBEATS-FiLM and AQ-NBEATS-OUT variants are significantly less sensitive to input normalization (Max-Norm) than the AQ-NBEATS-CAT variant. The main reason for this is that the time-series in the dataset have large magnitude. The quantile probability, distributed in $[0, 1]$ range, when concatenated with a vector of large magnitude values, has a tendency to be neglected. Based on these observations, we believe that AQ-NBEATS-FiLM and AQ-NBEATS-OUT variants should be preferred in

Table 4
Ablation study of AQ-ESRNN.

	CRPS	N-CRPS	MAPE
AQ-ESRNN	195.9	1.72	2.32
AQ-ESRNN, 4 Layers	198.7	1.75	2.42
AQ-ESRNN, 3 Layers	202.7	1.77	2.41
AQ-ESRNN-noQuantInMidLayers	203.7	1.78	2.37
AQ-ESRNN-noInputInMidLayers	204.2	1.79	2.41
AQ-ESRNN-dRNN	199.2	1.74	2.36
AQ-ESRNN-LSTM	204.0	1.79	2.41
AQ-ESRNN-unsorted	264.2	2.34	2.35

Table 5
Ablation study of the Max-Norm and the concatenation/FiLM based quantile incorporation mechanisms in the N-BEATS architecture.

	Max-Norm	MQN	CRPS	N-CRPS	MAPE
AQ-NBEATS-CAT	✓		215.3	1.94	2.62
AQ-NBEATS-CAT			286.7	2.58	2.57
AQ-NBEATS-FiLM	✓		219.3	1.98	2.68
AQ-NBEATS-FiLM			214.1	1.86	2.51
AQ-NBEATS-OUT	✓		215.7	1.94	2.61
AQ-NBEATS-OUT			217.2	1.9	2.56
AQ-NBEATS-CAT	✓	✓	215.5	1.86	2.52
AQ-NBEATS-CAT		✓	285.1	2.46	2.48
AQ-NBEATS-FiLM	✓	✓	228.1	1.94	2.62
AQ-NBEATS-FiLM		✓	211.2	1.84	2.47
AQ-NBEATS-OUT	✓	✓	216.4	1.86	2.52
AQ-NBEATS-OUT		✓	212.5	1.84	2.48

architecture designs due to robustness, accuracy and computational efficiency considerations. Indeed, recall that AQ-NBEATS-CAT deals with the $B \times T \times Q$ (batch, time, quantiles) input tensor due to the concatenation of quantiles, effectively multiplying the inference compute cost by a factor of Q . AQ-NBEATS-FiLM has similar compute explosion, but clearly better accuracy. AQ-NBEATS-OUT has comparable accuracy, but compute cost is multiplied by a factor of Q only for one of the blocks, making it order of magnitude more compute efficient. Finally, the normalized MQ loss defined in (17) may provide improvements, depending on architecture configurations, and can be considered as a hyperparameter to obtain incremental improvements in accuracy.

7. Discussion of findings

Flexibility of AQ-approach to Probabilistic Forecasting. The review of related work on probabilistic forecasting highlights diverse approaches in Section 1.1. These can be broadly categorized into (i) methods focusing on non-parametric quantile regression with fixed quantile probabilities and (ii) methods that adopt a parametric distribution models allowing them to predict the full distribution and thus arbitrary quantiles. Our methodology falls within the realm of non-parametric models. As such, it can model arbitrary conditional distributions. Yet, it retains the flexibility inherent to parametric techniques by being able to predict an arbitrary quantile. This is achieved through (i) the general parameterization of the forecaster $f_\theta(\mathbf{x}, q)$, treating the quantile probability q as an input parameter and (ii) the learning methodology that trains the model to forecast future trajectories based on input patterns $\mathbf{x}$ conditioned on randomly selected quantile probabilities q . Consequently, our AQ-models offer enhanced flexibility and accuracy compared to classical and deep probabilistic forecasting baselines, which is supported by our experimental results.

Quantile Injection Mechanism. We experimented with different mechanisms used to inject the quantile probability value in the network. Results for both ESRNN and N-BEATS architectures seem to indicate that the accuracy-optimal injection mechanism exposes every block of the deep neural network to the value of desired quantile probability. The application of quantile only at the input and output works for both ESRNN and N-BEATS, being marginally worse. Additionally,

both simple concatenation and more sophisticated mechanisms such as FiLM layer seem viable for injecting quantile probability. At the same time, concatenation works well in the case when input signal normalization is applied (which is the case of ESRNN by design and it can be used for N-BEATS additionally). FiLM works fine with both normalized and unnormalized data. Finally, injecting quantile probability at the last block of a deep learning architecture seems to be most computationally effective. Given that this is only marginally worse than injecting the quantile at every intermediate architectural block, this method can be preferable in applications requiring fast inference speed. We conjecture that this result implies that most of time-series representation work is independent of the quantile that the network targets. Therefore, the task of time-series representation and the task of quantile generation are largely disjoint and can be handled in a substantially independent way within the forecasting backbone, leading to inference speed gains.

Quantile Crossing and Sorting. In general, machine learning models trained for quantile regression may exhibit undesirable quantile crossing behavior, where a forecast for a lower quantile exceeds that of a higher quantile. While the frequency of these occurrences can be reduced by augmenting the loss function with a suitable penalty [65], this does not guarantee that quantile crossing will never occur. Achieving this requires a more complex model, as demonstrated in studies such as [65,66]. Alternatively, post-processing methods can be applied, such as sorting or isotonic projection [67]. Sorting is the simplest method, but it has been shown to have desirable properties, such as ensuring that CRPS never decreases [68]. Interestingly, the two models proposed in this paper, AQ-ESRNN and AQ-NBEATS, respond differently to sorting: it is a crucial step for the former (the metrics improve by 30%, see Table 4) and is practically unimportant for the latter. We leave the in-depth study of root-causes of this phenomenon for future work. In the current work, we conclude that, generally speaking, sorting is a powerful mechanism for improving multi-quantile forecasting, whereas its effectiveness may vary depending on underlying model and problem.

Probability Sampling Distribution. Another distinction between the two models lies in the sampling distribution utilized for generating probabilities during training. AQ-ESRNN employs Beta(0.3, 0.3) (please refer to Fig. 3), while AQ-NBEATS remains unaffected by this innovation and utilizes the uniform distribution. Originally, the idea of employing a convex Beta distribution for AQ-ESRNN was introduced to enhance the accuracy of extreme quantiles (where probability is close to 0 or 1), with the expectation that improvements in accuracy in these areas would negatively impact overall accuracy. However, it transpired that this approach also benefits the overall distribution metric, such as CRPS. We speculate that this difference between the two models stems from the fact that AQ-NBEATS employs ReLU nonlinearity, which allows the output to easily grow to any required value, while AQ-ESRNN employs sigmoid and tanh nonlinearities, which may pose more challenges for the output to reach extreme values.

Differences from Prior Work. While our method is related to [38, 69] there are also fundamental distinctions. We focus on cross time-series forecasting scenario, presenting the following novel contributions. First, we design advanced any-quantile forecasting architectures for two distinct classes of time-series backbones: ES-RNN and N-BEATS. We do not use an additional Implicit Quantile Network (IQN) to modify the output of a forecasting NN. Instead, we employ a single end-to-end NN that takes the sampled quantile level as one of its inputs. Importantly, our models, particularly those based on recurrent neural networks, explicitly capture temporal dependencies in the quantile space. This is achieved through built-in mechanisms such as hidden states, dilations, attention-based cells, and dynamic exponential smoothing. These features allow the models to learn both short- and long-term temporal patterns relevant for probabilistic forecasting, thereby extending beyond the scope of prior methods. We provide empirical evidence to support the effectiveness of the proposed architectural solutions and study the trade-offs between accuracy and computational complexity of

these approaches. Second, we provide empirical evidence of accuracy improvement of the proposed approaches compared to a large array of benchmarks including statistical models, quantile regression and NGBoost model operating in the local training and inference scenario (looking at one time-series at a time); as well as compared to an array of global approaches based on models from the GluonTS library. Finally, we present theoretical development linking cross-learning over a collection of time-series to the general information fusion problem.

Additional Data. While we recognize that meteorological conditions and holiday effects significantly influence electricity demand, our current models do not incorporate these exogenous factors. The primary limitation is data availability: specifically, the lack of archived hourly weather forecasts (up to 48 h ahead) spanning the entire 2006–2018 period across 35 European countries. Although real-time services such as open-meteo.com or yr.no provide current forecasts, to the best of our knowledge, they do not offer historical forecast archives suitable for retrospective modeling. Furthermore, even with access to such data, the inherent uncertainty in weather forecasts — particularly at longer horizons — could introduce additional noise into the learning and inference process. As such, the overall utility of incorporating these inputs remains an open question, which we consider a promising direction for future investigation. Regarding holiday effects, our models currently treat holidays as regular days, effectively acknowledging higher forecast uncertainty during these periods. Prior work has proposed dedicated modeling strategies for holiday corrections [70–72]. We view the integration of such corrections as a practical and modular extension, and leave this for future work.

8. Conclusion

We present a general probabilistic forecasting methodology applicable to a wide range of neural network forecasting algorithms. During training, this methodology involves exposing the forecasting algorithm to the task of predicting a randomly sampled quantile. Once trained, the algorithm is capable of providing any arbitrary quantile during inference. We applied this general methodology to two well-known neural forecasting algorithms through novel architectural modifications. Our approach leverages information fusion by employing a cross-learning strategy that combines data from multiple countries. This integration of diverse sources of uncertainty within power systems and complex regional patterns enhances forecasting performance. Empirically, we demonstrated that the proposed modified algorithms significantly outperform several classical and neural forecasting baselines.

Moving forward, our research will focus on extending our current results to additional neural network architectures and other machine learning models, as well as demonstrating the effectiveness of the proposed methodology across different application domains.

CRedit authorship contribution statement

Slawek Smyl: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Boris N. Oreshkin:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Paweł Pełka:** Validation, Software, Resources, Investigation. **Grzegorz Dudek:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Hyperparameters of the baseline models

Table A.6

Summary of classical forecasting models used in the study.

Model type	Parameterization/Notes
ARIMA	Automatic selection of (p, d, q) and (P, D, Q) via AICc minimization
ETS	Automatic configuration of Error, Trend, and Seasonal components (Additive, Multiplicative, Damped, or None)
Theta	Automatically optimized parameters of the Theta method, Dynamically tuned over time

Table A.7

Quantile regression hyperparameters.

Hyperparameter	Value/Search range
Solver	'highs'
fit_intercept	True
alpha	[0.0, 5.0]

Hyperparameter alpha was optimized for each country using Optuna with 100 trials.

Table A.8

NGBoost hyperparameters (default values with modifications).

Hyperparameter	Value
learning_rate	0.01 (default: 0.002)
n_estimators	100 (default: 500)
Base_learner	DecisionTreeRegressor
Minibatch_frac	1.0
Col_sample	1.0
Tol	1e−4
Validation_fraction	0.1
Early_stopping_rounds	None
Score	LogScore
Natural_gradient	True

Table A.9

Hyperparameters of baseline models from GluonTS (default values).

Common hyperparameters for all models:		
Hyperparameter	Value	
Epochs	100	
Batch size	32	
Optimizer	Adam	
Learning rate	0.001	
Dropout	0.1	
Model-specific hyperparameters:		
Model	Hyperparameter	Value
MLP	Size	[40,40]
	Activation	ReLU
DeepAR	Size	40
	Layers	2
	Cell type	LSTM
	Scaling	True
Transformer	Model dimension	32
	Inner FF dimension scale	4
	Number of heads	8
WaveNet	Residual channels	24
	Skip channels	32
	Number of stacks	1
	Epochs	200
	Learning rate	0.01
TFT	Size	32
	Variable selection	True
	Attention heads	4

Data availability

The real-world dataset utilized in this study is sourced from the ENTSO-E repository (www.entsoe.eu/data/power-stats). This dataset encompasses hourly electricity demand data for 35 European countries

and we have made this dataset publicly available along with the model code in our GitHub repository <https://github.com/boreshkinai/any-quantile>.

References

- [1] S.E. Haupt, M. Garcia Casado, M. Davidson, J. Dobschinski, P. Du, M. Lange, T. Miller, C. Mohrlen, A. Motley, R. Pestana, J. Zack, The use of probabilistic forecasts: Applying them in theory and practice, *IEEE Power Energy Mag.* 17 (6) (2019) 46–57, <http://dx.doi.org/10.1109/MPE.2019.2932639>.
- [2] P. Etingov, L. Miller, Z. Hou, Y. Makarov, K. Pennock, P. Beaucage, C. Loutan, A. Motley, Balancing needs assessment using advanced probabilistic forecasts, in: 2018 IEEE International Conference on Probabilistic Methods Applied To Power Systems, PMAPS, 2018, pp. 1–6, <http://dx.doi.org/10.1109/PMAPS.2018.8440392>.
- [3] Q. Wang, A. Tuohy, M. Ortega-Vazquez, M. Bello, E. Ela, D. Kirk-Davidoff, W.B. Hobbs, D.J. Ault, R. Philbrick, Quantifying the value of probabilistic forecasting for power system operation planning, *Appl. Energy* 343 (2023) 121254, <http://dx.doi.org/10.1016/j.apenergy.2023.121254>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261923006189>.
- [4] S. Yamujala, A. Jain, S. Sreekumar, R. Bhakar, J. Mathur, Enhancing power systems operational flexibility with ramp products from flexible resources, *Electr. Power Syst. Res.* 202 (2022) 107599, <http://dx.doi.org/10.1016/j.ejpsr.2021.107599>, URL: <https://www.sciencedirect.com/science/article/pii/S0378779621005800>.
- [5] D. Le, A. Berizzi, C. Bovo, A probabilistic security assessment approach to power systems with integrated wind resources, *Renew. Energy* 85 (2016) 114–123, <http://dx.doi.org/10.1016/j.renene.2015.06.035>, URL: <https://www.sciencedirect.com/science/article/pii/S0960148115300598>.
- [6] M. Beykirch, T. Janke, F. Steinke, Bidding and scheduling in energy markets: Which probabilistic forecast do we need? in: 2022 17th International Conference on Probabilistic Methods Applied To Power Systems, PMAPS, 2022, pp. 1–6, <http://dx.doi.org/10.1109/PMAPS53380.2022.9810632>.
- [7] T. Hong, S. Fan, Probabilistic electric load forecasting: A tutorial review, *Int. J. Forecast.* 32 (3) (2016) 914–938, <http://dx.doi.org/10.1016/j.ijforecast.2015.11.011>, URL: <https://www.sciencedirect.com/science/article/pii/S0169207015001508>.
- [8] H. Khajeh, H. Laaksonen, Applications of probabilistic forecasting in smart grids: A review, *Appl. Sci.* 12 (4) (2022) <http://dx.doi.org/10.3390/app12041823>, URL: <https://www.mdpi.com/2076-3417/12/4/1823>.
- [9] B. Li, J. Zhang, A review on the integration of probabilistic solar forecasting in power systems, *Sol. Energy* 210 (2020) 68–86, <http://dx.doi.org/10.1016/j.solener.2020.07.066>, Special Issue on Grid Integration. URL: <https://www.sciencedirect.com/science/article/pii/S0038092X20307982>.
- [10] R.J. Hyndman, S. Fan, Density forecasting for long-term peak electricity demand, *IEEE Trans. Power Syst.* 25 (2) (2010) 1142–1153, <http://dx.doi.org/10.1109/TPWRS.2009.2036017>.
- [11] J.W. Taylor, R. Buizza, Neural network load forecasting with weather ensemble predictions, *IEEE Trans. Power Syst.* 17 (3) (2002) 626–632, <http://dx.doi.org/10.1109/TPWRS.2002.800906>, URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0036699869&doi=10.1109%2fTPWRS.2002.800906&partnerID=40&md5=019f35fba591b6a47c11ac1e27c6fdb>.
- [12] A. Khoshrou, E.J. Pauwels, Short-term scenario-based probabilistic load forecasting: A data-driven approach, *Appl. Energy* 238 (2019) 1258–1268, <http://dx.doi.org/10.1016/j.apenergy.2019.01.155>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261919301412>.
- [13] J. Xie, T. Hong, Temperature scenario generation for probabilistic load forecasting, *IEEE Trans. Smart Grid* 9 (3) (2018) 1680–1687, <http://dx.doi.org/10.1109/TSG.2016.2597178>.
- [14] J. Xie, T. Hong, T. Laing, C. Kang, On normality assumption in residual simulation for probabilistic load forecasting, *IEEE Trans. Smart Grid* 8 (3) (2017) 1046–1053, <http://dx.doi.org/10.1109/TSG.2015.2447007>.
- [15] C.-L. Hor, S.J. Watson, S. Majithia, Daily load forecasting and maximum demand estimation using ARIMA and GARCH, in: 2006 International Conference on Probabilistic Methods Applied To Power Systems, 2006, pp. 1–6, <http://dx.doi.org/10.1109/PMAPS.2006.360237>.
- [16] Z. Cao, C. Wan, Z. Zhang, F. Li, Y. Song, Hybrid ensemble deep learning for deterministic and probabilistic low-voltage load forecasting, *IEEE Trans. Power Syst.* 35 (3) (2020) 1881–1897, <http://dx.doi.org/10.1109/TPWRS.2019.2946701>.
- [17] B. Liu, J. Nowotarski, T. Hong, R. Weron, Probabilistic load forecasting via quantile regression averaging on sister forecasts, *IEEE Trans. Smart Grid* 8 (2) (2017) 730–737, <http://dx.doi.org/10.1109/TSG.2015.2437877>.
- [18] Y. Yang, S. Li, W. Li, M. Qu, Power load probability density forecasting using Gaussian process quantile regression, *Appl. Energy* 213 (2018) 499–509, <http://dx.doi.org/10.1016/j.apenergy.2017.11.035>, URL: <https://www.sciencedirect.com/science/article/pii/S03062619171316100>.
- [19] Y. He, Y. Qin, S. Wang, X. Wang, C. Wang, Electricity consumption probability density forecasting method based on LASSO-quantile regression neural network, *Appl. Energy* 233–234 (2019) 565–575, <http://dx.doi.org/10.1016/j.apenergy.2018.10.061>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261918316301>.
- [20] R. Liu, T. Chen, G. Sun, S. Mueen, S. Lin, Y. Mi, Short-term probabilistic building load forecasting based on feature integrated artificial intelligent approach, *Electr. Power Syst. Res.* 206 (2022) 107802, <http://dx.doi.org/10.1016/j.ejpsr.2022.107802>, URL: <https://www.sciencedirect.com/science/article/pii/S0378779622000323>.
- [21] Y. He, C. Cao, S. Wang, H. Fu, Nonparametric probabilistic load forecasting based on quantile combination in electrical power systems, *Appl. Energy* 322 (2022) 119507, <http://dx.doi.org/10.1016/j.apenergy.2022.119507>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261922008273>.
- [22] B. Li, Y. Mo, F. Gao, X. Bai, Short-term probabilistic load forecasting method based on uncertainty estimation and deep learning model considering meteorological factors, *Electr. Power Syst. Res.* 225 (2023) 109804, <http://dx.doi.org/10.1016/j.ejpsr.2023.109804>, URL: <https://www.sciencedirect.com/science/article/pii/S0378779623006934>.
- [23] Y. He, R. Liu, H. Li, S. Wang, X. Lu, Short-term power load probability density forecasting method using Kernel-based support vector quantile regression and Copula theory, *Appl. Energy* 185 (2017) 254–266, <http://dx.doi.org/10.1016/j.apenergy.2016.10.079>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261916315239>.
- [24] Y. He, J. Xiao, X. An, C. Cao, J. Xiao, Short-term power load probability density forecasting based on GLRQ-stacking ensemble learning method, *Int. J. Electr. Power Energy Syst.* 142 (2022) 108243, <http://dx.doi.org/10.1016/j.ijepes.2022.108243>, URL: <https://www.sciencedirect.com/science/article/pii/S0142061522002721>.
- [25] G. Dudek, Stacking for probabilistic short-term load forecasting, in: L. Franco, et al. (Eds.), *Computational Science – ICCS 2024*, in: LNCS, vol. 14833, Springer, Cham, 2024, pp. 3–18, http://dx.doi.org/10.1007/978-3-031-63751-3_1.
- [26] T. Hong, P. Pinson, Y. Wang, R. Weron, D. Yang, H. Zareipour, Energy forecasting: A review and outlook, *IEEE Open Access J. Power Energy* 7 (2020) 376–388, <http://dx.doi.org/10.1109/OAJPE.2020.3029979>.
- [27] Y. Wang, D. Gan, M. Sun, N. Zhang, Z. Lu, C. Kang, Probabilistic individual load forecasting using pinball loss guided LSTM, *Appl. Energy* 235 (2019) 10–20, <http://dx.doi.org/10.1016/j.apenergy.2018.10.078>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261918316465>.
- [28] W. Zhang, H. Quan, D. Srinivasan, An improved quantile regression neural network for probabilistic load forecasting, *IEEE Trans. Smart Grid* 10 (4) (2019) 4425–4434, <http://dx.doi.org/10.1109/TSG.2018.2859749>.
- [29] D. Li, Y. Tan, Y. Zhang, S. Miao, S. He, Probabilistic forecasting method for mid-term hourly load time series based on an improved temporal fusion transformer model, *Int. J. Electr. Power Energy Syst.* 146 (2023) 108743, <http://dx.doi.org/10.1016/j.ijepes.2022.108743>, URL: <https://www.sciencedirect.com/science/article/pii/S0142061522007396>.
- [30] S. Smyl, G. Dudek, P. Pelka, ES-dRNN: A hybrid exponential smoothing and dilated recurrent neural network model for short-term load forecasting, *IEEE Transactions on Neural Networks and Learning Systems* 35 (8) (2024) 11346–11358, <http://dx.doi.org/10.1109/TNNLS.2023.3259149>.
- [31] L. Alfieri, P. De Falco, Wavelet-based decompositions in probabilistic load forecasting, *IEEE Trans. Smart Grid* 11 (2) (2020) 1367–1376, <http://dx.doi.org/10.1109/TSG.2019.2937072>.
- [32] D. van der Meer, M. Shepero, A. Svensson, J. Widén, J. Munkhammar, Probabilistic forecasting of electricity consumption, photovoltaic power generation and net demand of an individual building using Gaussian processes, *Appl. Energy* 213 (2018) 195–207, <http://dx.doi.org/10.1016/j.apenergy.2017.12.104>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261917318275>.
- [33] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D.C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, Gluonts: Probabilistic and neural time series modeling in python, *J. Mach. Learn. Res.* 21 (1) (2020) 4629–4634.
- [34] A. Brusaferrri, M. Matteucci, S. Spinelli, A. Vitali, Probabilistic electric load forecasting through Bayesian mixture density networks, *Appl. Energy* 309 (2022) 118341, <http://dx.doi.org/10.1016/j.apenergy.2021.118341>, URL: <https://www.sciencedirect.com/science/article/pii/S0306261921015907>.
- [35] S. Smyl, A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting, *Int. J. Forecast.* 36 (1) (2020) 75–85, <http://dx.doi.org/10.1016/j.ijforecast.2019.03.017>, M4 Competition. URL: <https://www.sciencedirect.com/science/article/pii/S0169207019301153>.
- [36] B.N. Oreshkin, D. Carpo, N. Chapados, Y. Bengio, N-BEATS: Neural basis expansion analysis for interpretable time series forecasting, in: *ICLR*, 2020.
- [37] B.N. Oreshkin, G. Dudek, P. Pelka, E. Turkina, N-BEATS neural network for mid-term electricity load forecasting, *Appl. Energy* 293 (2021) 116918, <http://dx.doi.org/10.1016/j.apenergy.2021.116918>.
- [38] A. Gouttes, K. Rasul, M. Koren, J. Stephan, T. Naghibi, Probabilistic time series forecasting with implicit quantile networks, 2021, [arXiv:2107.03743](https://arxiv.org/abs/2107.03743).

- [39] H. Karimi, J. Nutini, M. Schmidt, Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition, in: P. Frasconi, N. Landwehr, G. Manco, J. Vreeken (Eds.), *Machine Learning and Knowledge Discovery in Databases*, Springer International Publishing, Cham, 2016, pp. 795–811, http://dx.doi.org/10.1007/978-3-319-46128-1_50.
- [40] T. Gneiting, R. Ranjan, Comparing density forecasts using threshold-and quantile-weighted scoring rules, *J. Bus. Econom. Statist.* 29 (3) (2011) 411–422, <http://dx.doi.org/10.1198/jbes.2010.08110>.
- [41] S. Smyl, G. Dudek, P. Pelka, ES-dRNN with dynamic attention for short-term load forecasting, in: 2022 International Joint Conference on Neural Networks, IJCNN, 2022, pp. 1–8, <http://dx.doi.org/10.1109/IJCNN55064.2022.9889791>.
- [42] A.-A. Semenov, E. Spiliotis, S. Makridakis, V. Assimakopoulos, Investigating the accuracy of cross-learning time series forecasting methods, *Int. J. Forecast.* 37 (3) (2021) 1072–1084, <http://dx.doi.org/10.1016/j.ijforecast.2020.11.009>, URL: <https://www.sciencedirect.com/science/article/pii/S0169207020301850>.
- [43] H. Hewamalage, C. Bergmeir, K. Bandara, Global models for time series forecasting: A simulation study, *Pattern Recognit.* 124 (2022) 108441, <http://dx.doi.org/10.1016/j.patcog.2021.108441>, URL: <https://www.sciencedirect.com/science/article/pii/S0031320321006178>.
- [44] R. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice*, second ed., OTexts, Australia, 2018.
- [45] E. Perez, F. Strub, H. de Vries, V. Dumoulin, A.C. Courville, Film: Visual reasoning with a general conditioning layer, in: AAAI, 2018, <http://dx.doi.org/10.1609/aaai.v32i1.11671>.
- [46] T. Poplawski, G. Dudek, J. Lyp, Forecasting methods for balancing energy market in Poland, *Int. J. Electr. Power Energy Syst.* 65 (2015) 94–101, <http://dx.doi.org/10.1016/j.ijepes.2014.09.029>.
- [47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimeshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: *NeurIPS*, 2019, pp. 8024–8035.
- [48] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2015, [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [49] W. Falcon, The PyTorch Lightning team, PyTorch Lightning, 2019, <http://dx.doi.org/10.5281/zenodo.3828935>, URL: <https://github.com/Lightning-AI/lightning>.
- [50] G. Dudek, Pattern similarity-based methods for short-term load forecasting – Part 2: Models, *Appl. Soft Comput.* 36 (2015) 422–441, <http://dx.doi.org/10.1016/j.asoc.2015.07.035>.
- [51] G. Dudek, Short-term load forecasting using theta method, in: 14th Int. Conf. on Forecasting in Power Engineering 2018, E3S Web Conf., Vol. 84, 2019, <http://dx.doi.org/10.1051/e3sconf/20198401004>.
- [52] R. Koenker, K.F. Hallock, Quantile regression, *J. Econ. Perspect.* 15 (4) (2001) 143–156, <http://dx.doi.org/10.1257/jep.15.4.143>.
- [53] T. Duan, A. Avati, D.Y. Ding, K.K. Thai, S. Basu, A.Y. Ng, A. Schuler, NGBoost: Natural gradient boosting for probabilistic prediction, in: H.D. III, A. Singh (Eds.), *Proceedings of the 37th International Conference on Machine Learning, ICML*, in: *Proceedings of Machine Learning Research*, Vol. 119, PMLR, 2020, pp. 2690–2700, URL: <https://proceedings.mlr.press/v119/duan20a.html>.
- [54] G. Dudek, Pattern-based local linear regression models for short-term load forecasting, *Electr. Power Syst. Res.* 130 (2016) 139–147, <http://dx.doi.org/10.1016/j.epsr.2015.09.001>.
- [55] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: Probabilistic forecasting with autoregressive recurrent networks, *Int. J. Forecast.* 36 (3) (2020) 1181–1191, <http://dx.doi.org/10.1016/j.ijforecast.2019.07.001>.
- [56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, et al., Attention is all you need, in: *Proceedings of 31st Conference on Neural Information Processing Systems, NIPS 2017*, 2017, pp. 5998–6008.
- [57] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, WaveNet: A generative model for raw audio, 2016, [arXiv:1609.03499](https://arxiv.org/abs/1609.03499).
- [58] B. Lim, S. Arik, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting, *Int. J. Forecast.* 37 (4) (2021) 1748–1764, <http://dx.doi.org/10.1016/j.ijforecast.2021.03.012>.
- [59] R. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice*, third ed., OTexts, Melbourne, Australia, 2021, (Accessed 28 December 2023).
- [60] J.A. Fiorucci, T.R. Pellegrini, F. Louzada, F. Petropoulos, A.B. Koehler, Models for optimising the theta method and their relationship to state space models, *Int. J. Forecast.* 32 (4) (2016) 1151–1161, <http://dx.doi.org/10.1016/j.ijforecast.2016.02.005>.
- [61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, Édouard Duchesnay, Scikit-learn: Machine learning in Python, *J. Mach. Learn. Res.* 12 (85) (2011) 2825–2830, URL: <http://jmlr.org/papers/v12/pedregosa11a.html>.
- [62] F.X. Diebold, R.S. Mariano, Comparing predictive accuracy, *J. Bus. Econom. Statist.* 13 (3) (1995) 253–263, <http://dx.doi.org/10.1080/07350015.1995.10524599>.
- [63] S. Makridakis, E. Spiliotis, V. Assimakopoulos, Z. Chen, A. Gaba, I. Tsetlin, R.L. Winkler, The M5 uncertainty competition: Results, findings and conclusions, *Int. J. Forecast.* 38 (4) (2022) 1365–1385, <http://dx.doi.org/10.1016/j.ijforecast.2021.10.009>, Special Issue: M5 competition. URL: <https://www.sciencedirect.com/science/article/pii/S0169207021001722>.
- [64] T. Gneiting, A.E. Raftery, Strictly proper scoring rules, prediction, and estimation, *J. Amer. Statist. Assoc.* 102 (477) (2007) 359–378, <http://dx.doi.org/10.1198/016214506000001437>.
- [65] W. Tang, G. Shen, Y. Lin, J. Huang, Nonparametric quantile regression: Non-crossing constraints and conformal prediction, 2022, [arXiv:2210.10161](https://arxiv.org/abs/2210.10161).
- [66] Y. Park, D. Maddix, F.-X. Aubet, K. Kan, J. Gasthaus, Y. Wang, Learning quantile functions without quantile crossing for distribution-free time series forecasting, 2022, [arXiv:2111.06581](https://arxiv.org/abs/2111.06581).
- [67] R. Fakoor, T. Kim, J. Mueller, A.J. Smola, R.J. Tibshirani, Flexible model aggregation for quantile regression, 2023, [arXiv:2103.00083](https://arxiv.org/abs/2103.00083).
- [68] V. Chernozhukov, I. Fernández-Val, A. Galichon, Quantile and probability curves without crossing, *Econom.* 78 (3) (2010) 1093–1125, <http://dx.doi.org/10.3982/ecta7880>.
- [69] N. Tagasovska, D. Lopez-Paz, Single-model uncertainties for deep learning, in: *Proc NeurIPS*, 2019.
- [70] J. Son, J. Cha, H. Kim, Y.-M. Wi, Day-ahead short-term load forecasting for holidays based on modification of similar days' load profiles, *IEEE Access* 10 (2022) 17864–17880.
- [71] J. Gunawan, C.-Y. Huang, An extensible framework for short-term holiday load forecasting combining dynamic time warping and LSTM network, *IEEE Access* 9 (2021).
- [72] S. Arora, J. Taylor, Short-term load forecasting for special days using rule-based models: A case study for France, *European J. Oper. Res.* (2018).