energies

MDPI

Article

Predicting Energy Demand in Semi-Remote Arctic Locations

Odin Foldvik Eikeland [1], Filippo Maria Bianchi [2], Harry Apostoleris [3], Morten Hansen [4], Yu-Cheng Chiou [1] and Matteo Chiesa [1,2,*]

[1] Department of Physics and Technology, UiT the Arctic University of Norway, 9037 Tromsø, Norway; odin.f.eikeland@uit.no (O.F.E.); yu.cheng.chiou@uit.no (Y.-C.C.)
[2] Department of Mathematics and Statistics and NORCE, The Norwegian Research Centre, UiT the Arctic University of Norway, 9037 Tromsø, Norway; filippo.m.bianchi@uit.no
[3] Laboratory for Energy and NanoScience (LENS), Masdar Institute Campus, Khalifa University of Science and Technology, 127788 Abu Dhabi, UAE; harry.apostoleris@gmail.com
[4] Ishavskraft Power Company, 9024 Tromsø, Norway; morten.hansen@ishavskraft.no
* Correspondence: matteo.chiesa@uit.no

Supplementary material

All the models used are described briefly in this supplementary chapter, where relevant references that describe the models more comprehensive are cited. In addition, a short discussion of the different advantages and disadvantages between the statistical- and the machine-learning approaches are discussed. Then, the case study for the household sector, which follows the same methodology as for the industry case are described. In the end, all the hyperparameter configurations used for predictions are given in chapter 5.5

Citation: Eikeland, O.F.; Bianchi, F.M.; Apostoleris, H.; Hansen, M.; Chiou, Y.-C.; Chiesa, M. Predicting Energy Demand in Semi-Remote Arctic Locations. *Energies* **2021**, *14*, 798. https://doi.org/10.3390/en14040798

Received: 29 December 2020
Accepted: 27 January 2021
Published: 3 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

1. Statistical models

1.1. Autoregressive Integrated Moving Average (ARIMA)

The formalism ARIMA (p,d,q) can be used to define a large class of statistical models. The parameter p indicates the order of the auto-regressive component, d represents the initial differencing of the time series, and q the order of the moving average component. The value of the parameters must be carefully selected to achieve high prediction accuracies [1]. The equations for forecasting with ARIMA (p,q,d) are constructed as:

p: order of the autoregressive term
d: order of differentiation
q: order of the moving average term

$$\begin{aligned} y_t &= Y_t, && d = 0 \\ y_t &= Y_t - Y_{t-1}, && d = 1 \\ y_t &= Y_t - 2Y_{t-1} + Y_{t-2}, && d = 2 \end{aligned}$$

where y_t is the d^{th} difference of Y, which gives that the second difference of Y as the first difference of the first difference, i.e. the time series has performed first order differencing two times. This could be necessary if the time series are not stationary after performing differencing one time. The general forecasting equation for ARIMA predictions are:

$$\hat{y} = \mu + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} - \theta_1 e_{t-1} - \cdots \theta_q e_{t-q}, \quad (1)$$

Where θ is the moving average parameter, ϕ the slope coefficient, μ are the autoregressive constant and e are the exponential smoothing coefficient.

In order to determine the optimal order of the ARIMA model to be used, some statistical analyses and pre-processing are required. First, the autocorrelation function (ACF) and partial autocorrelation functions (PACF) are computed to get a better understanding of the data provided, and the ACF and PACF are interpreted to investigate whether the data is stationary or not. Stationarity in time series is when statistical properties such as the mean and the variance do not change over time during the observation period. When performing predictions with ARIMA models, the time series are required to be stationary. To check for stationarity, the ACF and PACF are plotted. If the plots show no correlations, the time series are stationary and ARIMA predictions could be performed. Finally, by interpreting the ACF and PACF plots, the correct orders of the AR and MA components are identified.

1.2. Prophet

The prophet-forecasting model developed by [2], uses a decomposable time series model divided into three main components [3]. The components are: Trend (h(t)), seasonality (s(t)) and holidays (h(t)). The trend function models non-periodic changes in the data and the seasonality represents periodic changes in the data (for instance daily, weekly or yearly seasonality). The holiday effect h(t) represent potentially irregular data over one or several days. The model components are added together as:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t, \quad (2)$$

Where ϵ_t are the error term representing changes that are not captured by the model, and are assumed to be normally distributed [2]. The trend are divided into two trend models, a saturating piecewise growth model and a piecewise linear model. The piecewise logistic growth model are derived to handle trend changes in the growth rate by explicitly defining change points where the growth rate are allowed to change. If the trend shows no saturating growth, the model are selected as a piecewise constant rate of growth. The seasonality it the time series are fitted by specifying seasonality models that are periodic functions of time. To capture periodic effects, the seasonality component rely on Fourier transformation that allows identifying the main periodicities in the time series that explain the consumption pattern [4]. The final component in the prophet-forecasting model, the holidays, are incorporated by assigning a dataset D_i that represent the set of past and future dates for the each holiday i. An indicator function are added, which represent whether time t is during the holiday i, and assign each holiday the parameter κ_i, which represent the corresponding change in the forecast.

When all the components are implemented in equation (2), the prophet-forecasting model in can be fitted to predict the specific task. All derivations of the different components are provided in detail in [2].

2. Neural networks

2.1. Elman Recurrent Neural Network (ERNN)

The ERNN, also known as the *Simple* RNN, is usually considered as the most basic version of RNN [5]. The more advanced RNN architectures such as GRU and LSTM can be interpreted as an extension of ERNN. The ERNN was proposed by Jeffrey L. Elman [6], where the aim was to generalize neural networks for better handling data sequences like time-series. The effectiveness of the RNNs in handling time series comes from the ability of learning of an input sequence by means of a recurrent function [7]. The layers in an ERNN are divided into; input, hidden and output layers. The input and output layers are characterized by feedforward connections, while the hidden layer contain recurrent connections. The specific ERNN processes one element of a sequence at time. At each timestamp t, the input layer process the information at $\mathbf{x}[t] \in \mathbb{R}^{N_i}$, where N_i are the number of nodes in the input layer. The input time series $\mathbf{x}$ has a total length T. In the input

layer, each component is summed with a bias vector $\mathbf{b_i} \in \mathbb{R}^{N_h}$, where N_h is the number of nodes in the hidden layer. Each component **x[t]** is then multiplied with a weight matrix $\mathbf{W}_i^h \in \mathbb{R}^{N_i \times N_h}$. Similarly, the internal state $\mathbf{h}[t-1] \in \mathbb{R}^{N_h}$ from the recurrent time interval is summarized with a bias vector $\mathbf{b_h} \in \mathbb{R}^{N_h}$, before multiplied with the weigh matrix $\mathbf{W}_h^h \in \mathbb{R}^{N_h \times N_h}$ of the recurrent connections. Then, the transformed input and past network state are combined and processed by the neurons in the hidden layers. Finally, the output of the network at timestamp t, are:

$$\mathbf{y}[t] = g\big(\mathbf{W}_h^o(\mathbf{h}[t] + \mathbf{b_o})\big), \quad (3)$$

where the output are computed through the transformation $g(\cdot)$ on the matrix of the output weights $\mathbf{W}_h^o \in \mathbb{R}^{N_r \times N_o}$ (N_o are number of nodes in the output layer). The output weights are applied to the sum of the current state **h[t]** and the bias vector $\boldsymbol{b_o} \in \mathbb{R}^{N_o}$ [5].

2.2. Long Short-Term Memory (LSTM)

The LSTM network are essentially build in a similar way as the ERNN architecture. The main difference between LSTM and ERNN is in the composition of the inner module, where the LSTM implements a more advanced internal processing unit, a cell [5,7].

The LSTM network has the same output and input as the original ERNN. However, internally it implements a gated system that controls the neural information. The key feature of the gated networks, which makes the LSTM as a widely used neural network, is the ability to try to solve the vanishing gradient problem by not imposing any bias towards recent observations. This provides the ability that the LSTM can maintain its internal memory unaltered for long time intervals [5,7].

2.3. Gated Recurrent Units (GRU)

The GRU networks are a simplified version of LSTM. The difference between GRU and LSTM network, is that in GRU, the forget and input gates are combined and merged into a single update. This controls how much each hidden unit can remember or forget. Therefore, the GRU network ends up having two gates compared to LSTM that has three gates [5,7]. Several works shows that GRUs can perform comparably to LSTM, but generally train faster due to lighter computation [8,9].

2.4. Convolutional Neural Network (CNN)

CNNs are a class of neural networks designed to work with data that can be structured in a grid-like topology [7,10]. CNNs has been widely used for image recognition and classification, but are also suitable for forecasting univariate time-series. The CNNs are based on a discrete convolution operator, consisting of an input vector **x,** kernel **w,** and an output **f.** The convolution operator produces the output by sliding the kernel over the input vector. Each element in the output feature is obtained by summing up the result of the element-wise multiplication between the input patch and the kernel. The number of kernels used in the convolutional layer determines the depth of the output volume. In this study, the CNN are applied on a univariate energy demand time-series $\mathbf{x} \in \mathbb{R}^{n_T}$**,** with a one-dimensional kernel $\mathbf{w} \in \mathbb{R}^{k}$. The output feature of the 1D CNN is:

$$f(i) = (\mathbf{x} \times \mathbf{w})(\mathbf{i}) = \sum_{j=0}^{k-1} x(i-j)\, w(j), \quad (4)$$

where i represents the i^{th} element of the convolution between **x** and **w** [7]. To better handle historical data, the original CNN are dilated to be able to learn long-term dependencies in the time-series. The dilated CNN proposed first time by the authors in [11], which named the dilated CNNs as Temporal Convolutional Network (TCN). In this study, three CNN layers are dilated by a dilation factor d, growing from d=1, d=2 and d=4. In each layer the kernel size is k=3.

The output from the TCN using the dilation factor d is:

$$f(i) = (\mathbf{x_d} \times \mathbf{w})(\mathbf{i}) = \sum_{j=0}^{k-1} x(i - dj)\, w(j), \quad (5)$$

All neural networks considered here (ERNN, LSTM, GRU and CNN) have been implemented with Python in Keras [12], with Tensorflow as backend [13].

2.5. Echo State Network (ESN)

The ESN consists of a large and untrained recurrent layer of nonlinear units in addition to a linear memory-less read-out layer. The full explanation of the underlying mechanisms for all components in the ESN architecture are explained in details in review paper by Bianchi et.al. [5].

To optimize predictions with ESN, the hyperparameter configuration must be tuned in order to maximize the prediction accuracy. Hyperparameters control the structure and the capacity of the model, determines how the network is trained, and must be specified before the ESN is trained to solve the prediction task [14]. In order for proper selection of hyperparameters to achieve accurate prediction results, the dataset analyzed are divided into three parts; training, validation and testing part [15,16].

The training set is used to fit the trainable parameters of the ESN, which are the weights of the readout. The generalization capability of the trained model, measured as the prediction accuracy obtained on future time steps of the energy load time series, is first evaluated on the validation set [15]. Since the weights of the ESN are not trained, the ESN performance is particularly sensitive to the choice of the hyperparameters.

Hyperparameters are selected to yield the highest mean accuracy on the validation set. Different strategies can be used to search for the optimal hyperparameters. Here, we tune the hyperparameters by performing a grid search over a large number of configurations (768), over 8 different hyperparameters. The hyperparameters that are optimized with cross-validation are; number of neurons in the reservoir (N_r), spectral radius (ρ), the regression parameters (C), noise in the state update (ξ), connectivity (R_c), and finally the scaling of the input, teaching and feedback weights ($\omega_i, \omega_o, \omega_f$).

When implementing and predicting the time series with ESN in this study, a modified version of the Python implementation provided by Løkse et al. is used [17].

3. Advantages and disadvantages of the statistical models and neural networks

A disadvantage of a model-based prediction approach such as the neural networks is that it is more difficult to interpret the results and understand the decision-process of the model. On the other hand, these models are easier to use for a practitioner and less domain-knowledge is required to use the models and obtain good results in terms of prediction. Statistical models are easier to interpret than the more advanced neural networks, and the user has more control on the procedure that generates the result. As a downside, a careful tuning that implies pre-analyses and a discrete amount of knowledge on the problem is required to achieve prediction results with high accuracy. This is in contrast to the neural networks, where no additional work except from careful tuning the hyperparameters is required when making predictions.

4. The household time series in location 1

This part of the supplementary chapter provides the same methodology as for the industry sector, this time for the aggregated household sector during workdays.

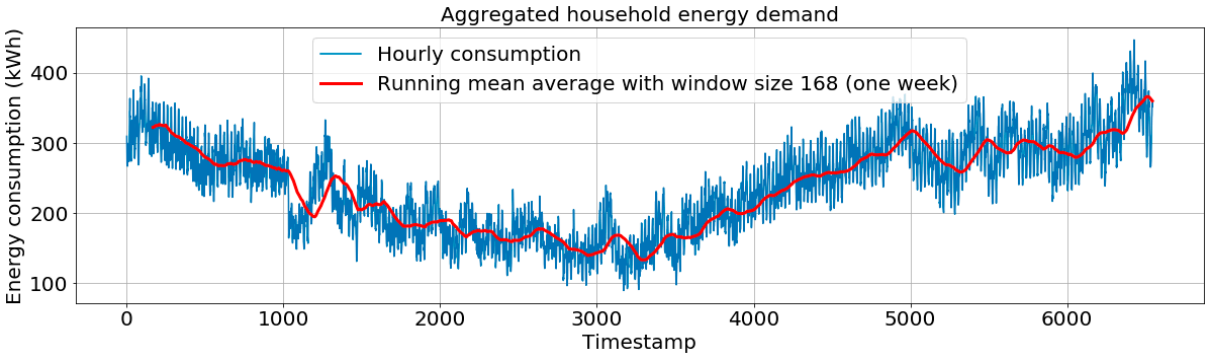

Figure S1. Energy demand for the aggregated household sector.

From Figure S1, the cyclic pattern is no longer easy to identify as for the industry sector. There is no strong weekly consumption pattern, and the consumption is much more frequent than every week as for the industry sector. During the summer period, the consumption is low as the weather is warm. The running mean average shows that the energy use evolves from high consumption early in July towards lower consumption in August. In the end of August, the consumption seems to increase again (from timestamp 3000 to 4000). The representation of main consumption pattern together with the ACF and PACF functions for the household sector are shown in Figure S2.

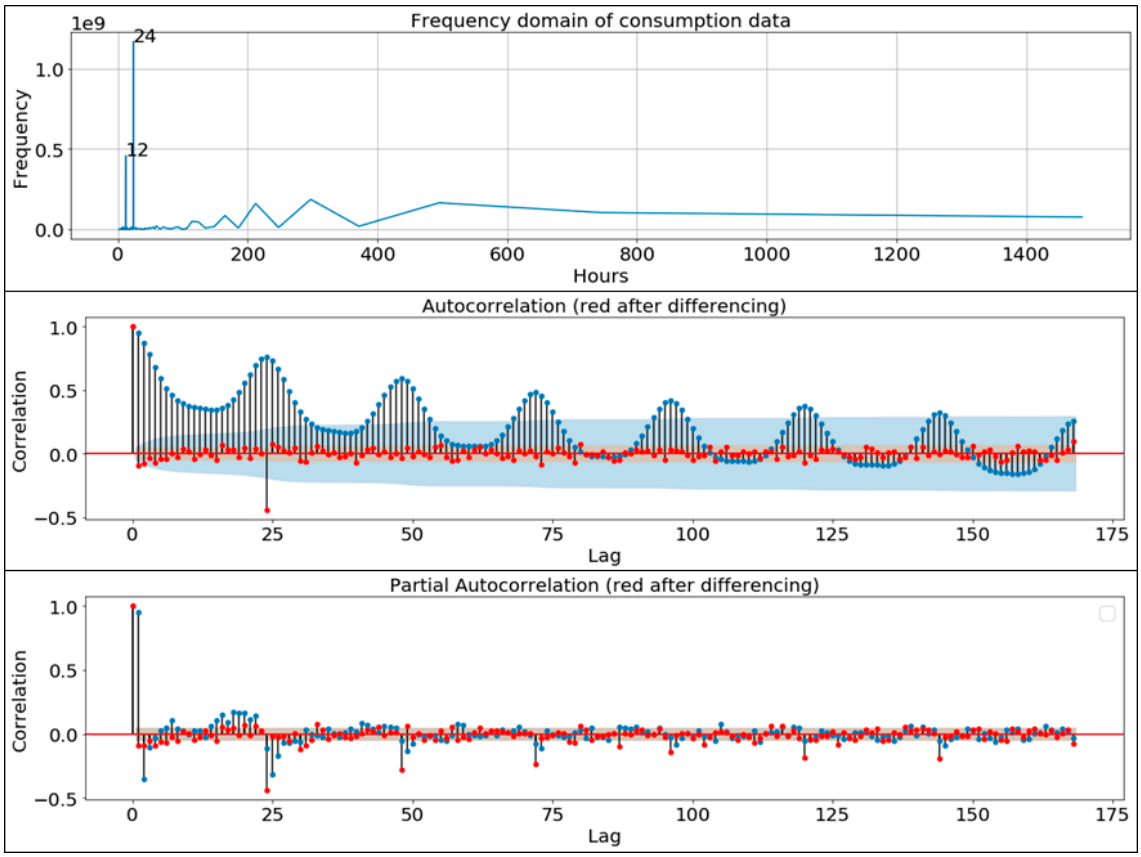

Figure S2. ACF and PACF plots for the energy consumption in households' sector. The correlation outside the standard deviations are correlations and not a statistical fluke.

The autocorrelation and partial autocorrelation show strong correlations outside the 95% confidence interval at approximately 12, 24 and 48 hours.

The Fourier transformation shows likewise a main seasonality at 24 hours. However, there is no longer a strong seasonality at 165 hours as for the industry sector. The consumption pattern is strongly driven by the daytime consumption pattern occurring every 12 hours, which in turn correlates well with the typical household daily life with routines as breakfast every morning and dinner every afternoon. The seasonality of the time series was again removed with seasonal differencing and thereafter the short-term correlation was removed by first order differencing and the ACF and PACF functions were plotted again, indicated by red colors in Figure S2.

5. Hyperparameter configurations and training of the models

For each model, the optimal hyperparameter configuration is searched on the validation set. In particular, we select as the optimal model the one yielding the highest prediction accuracy on the validation set. Then, the performance of the optimal model is evaluated on the test set.

5.1. ARIMA

To perform predictions with the ARIMA model, the time series need to be stationary. The seasonality of the time series is removed with seasonal differencing and thereafter the short-term correlation was removed by first order differencing. To check if the time series become stationary after differentiation, the ACF and PACF functions are plotted again, and they are depicted in red in Figure S2. As there are no longer strong correlations in the ACF and PACF plot (all values are almost zero in the red figure), we can assume that the time series are now stationary and ARIMA model can be applied. As the PACF function has no correlations, it is possible to use an ARIMA (1,0,1) predictive model.

In addition to selecting the ARIMA orders by interpreting the ACF and PACF plots, a grid search of different ARIMA orders was performed to identify the configuration that yields the highest prediction accuracy. The different ARIMA orders searched are provided in Table S1. Even according to this second optimization scheme, we found that the ARIMA (1,0,1) model achieved the highest prediction accuracy in terms of NRMSE.

Table S1. Each hyperparameter is searched in the interval [min,max]. The parameters in Table S1 are the following: Order of the autoregressive term (p), order of differentiation (d), and order of moving average term (p). The optimum hyperparameter configurations for each sector are selected as the one yielding the highest prediction accuracy on the validation set. The ARIMA(1,0,1) configuration are the model providing highest prediction accuracy for both industry and household sector.

ARIMA (p,d,q)	p	d	q
min	0	0	0
max	2	2	2
Optimal	1	0	1

To check for stationarity, the ACF and PACF are given for the residual errors (predicted value subtracted from the actual value). The ACF and PACF plots for the residual errors are provided in Figure S3.

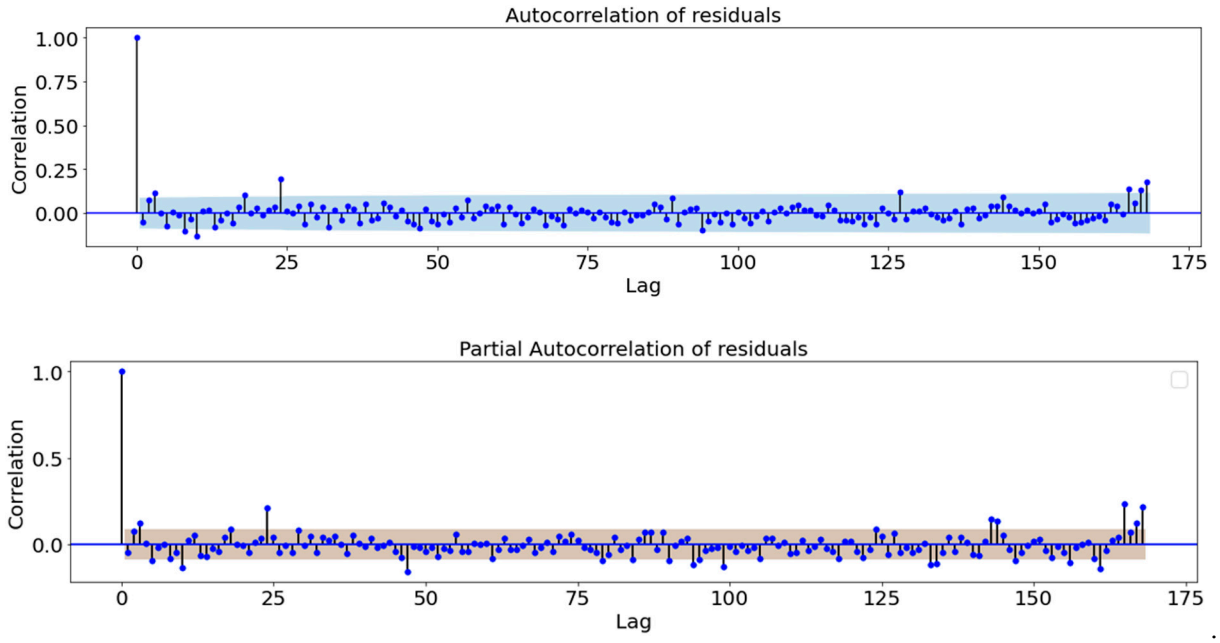

Figure S3. ACF and PACF plots for the residuals after predictions with the ARIMA(1,0,1) model.

The ACF and PACF of the residuals does not show any significant correlations. This suggests that the ARIMA (1,0,1) model is able to predict the time series.

5.2. Prophet

When making predictions with the open-source Prophet library, only the default configurations were used. The GitHub repository for making forecasts with Prophet could be found here [18]. The Prophet prediction method are developed and explained in detail in [2].

5.3. Trainable RNNs

The trainable RNNs (LSTM, GRU and Elman) are trained over several epochs on a set of configurations, where the epoch resulting in the highest accuracy on the validation set are selected for predictions on the test set. The hyperparameter configurations for the trainable RNNs are specified in Table S2.

Table S2. The hyperparameter configuration for the trainable RNNs are specified as: Number of layers (L), number of units per layer (n). The hyperparameter configuration are trained over 50 epochs.

	LSTM	**GRU**	**ERNN**
L	1	1	1
n	32	32	32
optimizer	adam	adam	adam
Learning rate	0.001	0.001	0.001
epochs	50	50	50

5.4. CNN

The CNN are trained in a similar way as for the trainable RNNs. However, inspired by the authors behind the Temporal Convolutional Network [11], we dilate the CNN to make the model able to learn long-term dependencies in the time-series. In this study, three CNN layers are dilated by a dilation factor d, growing from d=1, d=2 and d=4. In each layer, the kernel size is k=3. The hyperparameter configuration for the CNN are given in Table S3.

Table S3. The hyperparameter configuration for the CNN are: Number of layers (L), number of units per layer (n), the convolutional kernel size (k). The dilation rate d_r, specifies how each convolutional layer L, are dilated with a factor a factor 2^i. Here i are the specific layer of the network. The hyperparameter configuration are trained over 50 epochs.

	L	**n**	**k**	**d_r(L1,L2,L3)**	**optimizer**	**Learning rate**	**epochs**
CNN configurations	3	32	3	1,2,4	adam	0.001	50

5.5. ESN

When performing the predictions with ESN, the optimal hyperparameter configuration was selected with a grid search. Each hyperparameter was searched in the range specified in Table S4.

Table S4. Each hyperparameter is searched in the interval [min,max]. The parameters in the table are the following: Neurons in the reservoir (N_r), connectivity (R_c), noise in the state uptdate (ξ), spectral radius (ρ), the scaling of input, teaching and feedback weights (ω_i,ω_o,ω_f), and regression parameter C. The optimum hyperparameter configurations for each sector are selected as the one yielding the highest prediction accuracy on the validation set.

Hyperparameters	N_r	R_c	ξ	ρ	ω_i	ω_o	ω_f	C
min	300	0.15	0.0	0.5	0.1	0.25	0.0	0.001
max	500	0.45	0.01	1.0	0.4	1.0	0.1	1.0

Optimum industry location 1	300	0.15	0.0	0.9	0.1	1.0	0.0	0.001
Optimum household location 1	500	0.15	0.0	1.0	0.1	1.0	0.0	0.001

To optimize predictions with the trainable RNNs and the CNN, the optimal hyperparameter configuration could be selected by performing a grid search as for the ARIMA and ESN model. However, since we use a small number of units (32 vs the 300-500 used in the ESN model), the models are not prone to overfitting and, therefore, regularization is not required.

Additionally, since the parameters are optimized with gradient descent, the sensitivity on the hyperparameters is lower than for the ESN. Indeed, the ESN trade the precision of the gradient descent optimization with the redundancy of a large random reservoir that, inevitably, makes the model more sensitive to hyperparameters configuration.

For this reason, we can expect good performance from ERNN, LSTM, GRU, and CNN by using a fixed hyperparameters configuration.

References:

1. Nau, R. Introduction to ARIMA: Nonseasonal Models. Available online: https://people.duke.edu/~rnau/411arim.htm (accessed on 6 May 2020).
2. Taylor, S.J.; Letham, B. Forecasting at Scale. Am. Stat. 2018, 72, 37–45, doi:10.1080/00031305.2017.1380080.
3. Harvey, A.; Peters, S. Estimation procedures for structural time series models. J. Forecast. 1990, 9, 89–108
4. Omar, K. Deconstructing Time Series Using Fourier Transform, Medium. Available online: https://medium.com/@khairulomar/deconstructing-time-series-using-fourier-transform-e52dd535a44e (accessed on 9 July 2020).
5. Bianchi, F.M. Maiorino, E.; Kampffmeyer, M.C.; Rizzi, A.; Jenssen, R. An Overview and Comparative Analysis of Recurrent Neural Networks for Short Term Load Forecasting; Cornell University, Ithaca, New York: 2018, doi: arXiv: 1705.04378.
6. Elman, J.L. Finding structure in time. Cogn. Sci. 1990, 14, 179–211.
7. Gasparin, A.; Lukovic, S.; Alippi, C. Deep Learning for Time Series Forecasting: The Electric Load Case; Cornell University, Ithaca, New york: 2019
8. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; Cornell University: Ithaca, NY, USA, 2014. doi: arXiv: 1412.3555
9. Yin, W.; Kann, K.; Yu, M.; Schütze, H. Comparative Study of CNN and RNN for Natural Language Processing; Cornell University: Ithaca, NY, USA, 2017. doi: arXiv: 1702.01923
10. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE 1998, 86, 2278–2324.
11. Bai, S.; Kolter, J.; Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling; Cornell University: Ithaca, NY, USA, 2018. doi:arXiv: 1803.01271 .
12. Chollet, F.; et al. Keras, Github. Available online: https://github.com/fchollet/keras (accessed on 10 October 2020).
13. TensorFlow. An End-to-End Open Source Machine Learning Platform. Available online: https://www.tensorflow.org/ (accessed on 27 August 2020).
14. Radhakrishnan, P. What are Hyperparameters? and How to tune the Hyperparameters in a Deep Neural Network? Towards Data Science. Available online: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a (accessed on 9 July 2020).
15. Chollet, F. Section 4.2. Evaluating Machine-Learning Models. Available online: https://livebook.manning.com/book/deep-learning-with-python/chapter-4/43 (accessed on 9 July 2020).
16. Stewart, M. Predicting Stock Prices with Echo State Networks. Towards Data Science. Available online: https://towardsdatascience.com/predicting-stock-prices-with-echo-state-networks-f910809d23d4 (accessed on 20 July 2020).
17. Løkse, S; PythonESN, GitHub. Available online: https://github.com/siloekse/PythonESN (accessed on 21 April 2020).
18. Sean, J.T and Letham, B. Forecasting at Scale—Prophet. Facebook. Available online: https://facebook.github.io/prophet/ (accessed on 15 October 2020).