Received: 28 December 2020 | Revised: 18 May 2021 | Accepted: 5 July 2021

DOI: 10.1049/cit2.12060

CAAI Transactions on Intelligence Technology

REVIEW

Deep learning for time series forecasting: The electric load case

Alberto Gasparin[1] | Slobodan Lukovic[1] | Cesare Alippi[1,2]

[1]Faculty of Informatics, Università della Svizzera Italiana, Lugano, Switzerland

[2]Department of Electronics, Information, and Bioengineering, Politecnico di Milano, Milan, Italy

Correspondence

Alberto Gasparin, Faculty of Informatics, Università della Svizzera Italiana, 6900 Lugano, Switzerland.
Email: alberto.gasparin@usi.ch

Funding information

Innosuisse – Schweizerische Agentur für Innovationsförderung, Grant/Award Number: 1155002544

Abstract

Management and efficient operations in critical infrastructures such as smart grids take huge advantage of accurate power load forecasting, which, due to its non-linear nature, remains a challenging task. Recently, deep learning has emerged in the machine learning field achieving impressive performance in a vast range of tasks, from image classification to machine translation. Applications of deep learning models to the electric load forecasting problem are gaining interest among researchers as well as the industry, but a comprehensive and sound comparison among different—also traditional—architectures is not yet available in the literature. This work aims at filling the gap by reviewing and experimentally evaluating four real world datasets on the most recent trends in electric load forecasting, by contrasting deep learning architectures on short-term forecast (one-day-ahead prediction). Specifically, the focus is on feedforward and recurrent neural networks, sequence-to-sequence models and temporal convolutional neural networks along with architectural variants, which are known in the signal processing community but are novel to the load forecasting one.

KEYWORDS

deep learning, electric load forecasting, multi-step ahead forecasting, smart grid, time-series prediction

1 | INTRODUCTION

Smart grids aim at creating automated and efficient energy delivery networks that improve power delivery reliability and quality, along with network security, energy efficiency, and demand-side management aspects [1]. Modern power distribution systems are supported by advanced monitoring infrastructures that produce immense amount of data, thus enabling fine grained analytics and improved forecasting performance. In particular, electric load forecasting emerges as a critical task in the energy field, as it is of central relevance for a number of power distributions tasks at different levels of the grid. At the single-household level, accurate load forecasting can create savings opportunities while also help reduce the energy footprint. At a higher level in the grid (i.e., looking at the aggregated load from multiple households or buildings in general) electric load forecasting is a fundamental tool for power system operators as it provides support for different decision-making tasks such as the definition of better pricing strategies, the reduction of maintenance cost, improved demand-side management and efficient electrical energy storage management. Load forecasting is carried out at different time horizons, ranging from milliseconds to years, depending on the specific problem at hand.

Our main goal is to concisely review and assess the most appropriate deep learning models that could be utilised in the smart grid field specifically for load forecasting. In this work, we focus on the day-ahead prediction problem also referred to in the literature as *short-term load forecasting* (STLF) [2]. Since deregulation of electric energy distribution and wide adoption of renewables strongly affects daily market prices, STLF emerges to be of fundamental importance for efficient power supply [3]. Furthermore, we differentiate forecasting on the granularity level at which it is applied. For instance, in an individual household scenario, load prediction is a rather difficult task as power consumption patterns are highly volatile. Thus, despite not being a very relevant task from the perspective of smart grids, we consider it for the sake of testing the proposed models with challenging and noisy dynamics. On the contrary, aggregated load consumption, that is, that associated with a neighbourhood, a region, or even an entire state, is normally easier to predict as the resulting signal exhibits slower dynamics. Still, the problem remains extremely relevant for the industry as even a small—statistically sound—increase in prediction accuracy can be translated into significant savings.

Historical power loads are time series affected by several external time variant factors, such as weather conditions, human activities, type of industrial processes, temporal and seasonal characteristics, which make their predictions a challenging problem. A large variety of prediction methods has been proposed for electric load forecasting over the years, and only the most relevant ones are reviewed in this survey. Autoregressive moving average models (ARMA) were among the first model families used in short-term load forecasting [4, 5]. Soon they were replaced by autoregressive-integrated moving average (ARIMA) and seasonal ARIMA models [6] to cope with time variance often exhibited by load profiles. In order to include exogenous variables like temperature into the forecasting method, the autoregressive moving average model with eXogenous inputs (ARMAX) [7, 8] and the autoregressive-integrated moving average model with eXogenous inputs (ARIMAX) [9] were introduced. The main shortcoming of these system identification families is the linearity assumption for the system being observed, a hypothesis that does not generally hold. In order to solve this limitation, non-linear models like feed-forward neural networks were proposed and became attractive for those scenarios exhibiting significant non-linearity, as in load forecasting tasks [3, 10–13]. The intrinsic sequential nature of time series data was then exploited by considering sophisticated techniques ranging from advanced feed-forward architecture with residual connections [14] to convolutional approaches [15, 16] and recurrent neural networks [17, 18] along with their many variants such as the echo-state network [18–20], long short-term memory [18, 21–23] and gated recurrent unit [18, 24]. Moreover, some hybrid architectures have also been proposed, aiming to capture the temporal dependencies in the data with recurrent networks while performing a more general feature extraction operation with convolutional layers [25, 26].

Different surveys address the load forecasting topic by means of (not necessarily deep) neural networks. In [42], the authors focus on the use of some deep learning architectures for load forecasting. However, this review lacks a comprehensive comparative study of performance verified on common load forecasting benchmarks. The absence of valid cost-performance metric does not allow the report to make conclusive statements. In [18], an exhaustive overview of recurrent neural networks for short-term load forecasting is presented. The very detailed work considers one-layer (not deep) recurrent networks only. A comprehensive summary of the most relevant research dealing with short-term load forecasting (STLF) employing recurrent neural networks, convolutional neural networks and seq2seq models is presented in Table 1. It emerges that most of the works have been performed on different datasets, making it rather difficult—if not impossible—to assess their absolute performance and, consequently, recommend the best state of the art solutions for load forecasting.

In this survey, we consider the most relevant—and recent—deep architectures and contrast them—also not on deep models—in terms of performance accuracy on open-source benchmarks. The considered architectures include linear models, shallow and deep feed-forward neural networks, recurrent neural networks, sequence-to-sequence models and temporal convolutional neural networks. The experimental comparison is performed on four different real world datasets that are representatives of two distinct scenarios. Three datasets consider power consumption at an individual household level with a signal characterised by high-frequency components while the last dataset takes into account aggregation of several consumers.

Our contributions consist in the following:

- A comprehensive review. The survey provides a comprehensive investigation of deep learning architectures known to the smart grid literature as well as novel recent ones suitable for electric load forecasting.
- A multi-step prediction strategy comparison for recurrent neural networks: we study and compare how different prediction strategies can be applied to recurrent neural networks. To the best of our knowledge, this work has not been done yet for deep recurrent neural networks.
- A relevant performance assessment. To the best of our knowledge, the present work provides the first systematic experimental comparison of the most relevant deep learning architectures for the electric load forecasting problems of individual and aggregated electric demand. It should be noted that the envisaged architectures are domain-independent and, as such, can be applied in different forecasting scenarios. In order to make the experiments reproducible—a very important aspect not rarely underestimated—all datasets used in this survey, as well as the source code, are publicly available.

The rest of this work is organised as follows:

In Section 2, we formalise the forecasting problem along with the notation that will be used in this work. In Section 3, we introduce feed-forward neural networks (FNNs) and the main concepts relevant to the learning task. We also provide a short review of the literature regarding the use of FNNs for the load forecasting problem.

In Section 4, we sketch recurrent neural networks (RNNs) and overview their most advanced architectures: long short-term memory and gated recurrent unit networks.

In Section 5, sequence-to-sequence architectures (seq2seq) are discussed as a general improvement over recurrent neural networks. We present both, simple and advanced models built on the sequence-to-sequence paradigm.

In Section 6, convolutional neural networks are introduced, and one of their most recent variants, the temporal convolutional network (TCN), is presented as the state-of-the-art method for univariate time series prediction.

In Section 7, the real world datasets used for the models' comparison are presented. For each dataset, we provide a

TABLE 1 A summary of prior works that address the topic of electric load forecasting with deep learning models

Reference	Predictive family of models	Time horizon	Exogenous variables	Dataset (location)
[18]	LSTM, GRU, ERNN, NARX, ESN	D	-	Rome, Italy
[18]	LSTM, GRU, ERNN, NARX, ESN	D	T	New England [27]
[28]	ERNN	H	T, H, P, other*	Palermo, Italy
[17]	ERNN	H	T, W, H	Hubli, India
[20]	ESN	15 min to 1Y	-	Sceaux, France [29]
[21]	LSTM, NARX	D	-	Unknown
[22]	LSTM	D(?)	C, TI	Australia [30]
[23]	LSTM	2W to 4M	T, W, H, C, TI	France
[31]	LSTM	2D	T, P, H, C, TI	Unknown [32]
[24]	GRU	D	T, C, other**	Dongguan, China
[33]	LSTM, seq2seq	60 H	C, TI	Sceaux, France [29]
[34]	LSTM, seq2seq	12 H	T, C, TI	New England [35]
[36]	seq2seq	D	-	USA
[37]	seq2seq	0.5H, 4H, 10H	T, W, H, C	Unknown
[38]	seq2seq + attention	1H, 4H, 10H, 1D	-	Unknown
[15]	CNN	D	C, TI	USA
[16]	CNN	D	C, TI	Sceaux, France
[25]	CNN + LSTM	D	T, C, TI	North-China
[26]	CNN + LSTM	D	-	North-Italy
[39]	WaveNet-like	D	-	France
[40]	TCN	D	-	Portugal [41]

Abbreviations: CNN, Convolutional neural networks; ERNN, Elmann recurrent neural networks; GRU, Gated recurrent units; LSTM, long short-term memory; TCN, temporal convolutional network.

Dataset: the data source; a link is provided whenever available.

Time Horizon: H (hour), D (day), W (week), M (month), Y (year), ? (Not explicitly stated, thus, inferred from the text) *Exogenous variables:* T (temperature), W (wind speed), H (humidity), P (pressure), C (calendar including date and holidays information), TI (time).

*Other input features were created for this dataset, **categorical weather information is used (e.g., sunny, cloudy).

description of the preprocessing operations and the techniques that have been used to validate the models' performance.

Finally, In Section 8, we draw conclusions based on performance.

2 | PROBLEM DESCRIPTION

In basic multi-step-ahead electric load forecasting, a univariate time series $\mathbf{s} = [s[0], s[1] \ldots, s[T]]$ that spans through several years is given. Input data are presented to the different predictive families of models as a regressor vector composed of fixed time-lagged data associated with a window size of length n_T, which slides over the time series. The size of the time window is a hyperparameter whose optimal value has to be identified on the data at hand. Given this fixed length view of past values, a predictor f aims at forecasting the next n_O values of the time series. In this work, the forecasting problem is cast into a supervised learning problem. As such, given the input vector at discrete time t $\mathbf{x}_t = [s[t - n_T + 1], \ldots, s[t]] \in \mathrm{IR}^{n_T}$, the forecasting problem requires to infer the next n_O measurements $\mathbf{y_t} = [s[t + 1], \ldots, s[t + n_O]] \in \mathrm{IR}^{n_O}$ or a subset of it. To ease the notation, we express the input and output vectors in the reference system of the time window (relative time) instead of the time series one (absolute time). By following this approach, the input vector at discrete time t becomes $\mathbf{x}_t = [x_t[0], \ldots, x_t[n_T - 1]] \in \mathrm{IR}^{n_T}, x_t[i] = s[i + 1 + t - n_T]$ and the corresponding output vector is $\mathbf{y_t} = [y_t[n_T - 1], \ldots, y_t[n_T + n_O - 2]] \in \mathrm{IR}^{n_O}$. y_t characterises the real output values defined as $y_t[t] = x_t[t + 1], \quad \forall t \in T$. Similarly, we denote as $\hat{\mathbf{y}}_\mathbf{t} = f(\mathbf{x_t}; \hat{\boldsymbol{\Theta}}) \in \mathrm{IR}^{n_O}$, the prediction vector provided by a predictive model f whose parameters vector $\boldsymbol{\Theta}$ has been estimated by optimising a performance function.

Without loss of generality, in the remainder of the work, we drop the subscript t from the inner elements of $\mathbf{x_t}$ and $\mathbf{y_t}$. The introduced notation, along with the sliding window approach, is depicted in Figure 1.

In certain applications, we will additionally be provided with extra $d-1$ exogenous variables (e.g., the temperatures), each of which represent a univariate time series aligned in time with the data of electricity demand. In this scenario, the components of the regressor vector become vectors, that is, $\mathbf{x_t} = [\mathbf{x}[0], \ldots, \mathbf{x}[n_T - 1]] \in \mathrm{IR}^{n_T \times d}$. Indeed, each element of the input sequence is represented as $\mathbf{x}[t] = [x[t], z_0[t], \ldots, z_{d-2}[t]] \in \mathrm{IR}^d$, where $x[t] \in \mathrm{IR}$ is the scalar load measurement at time t, while $z_k[t] \in \mathrm{IR}$ is the scalar value of the k^{th} exogenous feature.

The nomenclature used in this work is given in Table 2.

3 | FEED-FORWARD NEURAL NETWORKS

Feed-forward neural networks (FNNs) are parametric model families characterised by the universal function approximation property [43]. The computational architectures are composed of a layered structure consisting of three main building blocks: the input layer, the hidden layer(s) and the output layer. The number of hidden layers ($L > 1$), determines the depth of the network, while the size of each layer, that is, the number $n_{H,\ell}$ of hidden units of the $\ell -$ th layer defines its complexity in terms of neurons. FNNs provide only direct forward connections between two consecutive layers, each connection associated with a trainable parameter; note that given the feed foward nature of the computation, no recursive feedback is allowed (as it happens instead in recurrent networks). More in detail, given a vector $\mathbf{x} \in \mathrm{IR}^{n_T}$ fed at the network input, the FNN's computation can be expressed as

$$\mathbf{a}_\ell = \mathbf{W}_\ell^{\mathrm{T}} \mathbf{h}_{\ell-1} + \mathbf{b}_\ell, \quad \ell = 1, \ldots L \tag{1}$$

$$\mathbf{h}_\ell = \phi_\ell(\mathbf{a}_\ell) \tag{2}$$

where $\mathbf{h}_0 = \mathbf{x_t} \in \mathrm{IR}^{n_T}$ and $\hat{\mathbf{y}}_\mathbf{t} = \mathbf{h}_L \in \mathrm{IR}^{n_O}$.

Each layer ℓ is characterised by its own parameters' matrix $\mathbf{W}_\ell \in \mathrm{IR}^{n_{H,\ell-1} \times n_{H,\ell}}$ and bias vector $\mathbf{b}_\ell \in \mathrm{IR}^{n_{H,\ell}}$. Hereafter, in order to ease the notation, we incorporate the bias term in the weight matrix, that is, $\mathbf{W}_\ell = [\mathbf{W}_\ell; \mathbf{b}_\ell]$ and $\mathbf{h}_\ell = [\mathbf{h}_\ell; 1]$. $\mathbf{\Theta} = [\mathbf{W}_1, \ldots, \mathbf{W}_L]$ groups all the network's parameters.

Given a training set of N input–output vectors in the ($\mathbf{x_i}$, $\mathbf{y_i}$) form, $i = 1, \ldots, N$, the learning procedure aims at identifying a suitable configuration of parameters $\hat{\mathbf{\Theta}}$ that minimises a loss function $\mathcal{L}(\mathbf{\Theta})$ evaluating the discrepancy between the estimated values $f(\mathbf{x_t}; \mathbf{\Theta})$ and the measurement $\mathbf{y_t}$:

$$\hat{\mathbf{\Theta}} = \arg\min_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta})$$

The mean squared error,

$$\mathcal{L}(\mathbf{\Theta}) = \frac{1}{N} \sum_{t=1}^{N} (\mathbf{y_t} - f(\mathbf{x_t}; \mathbf{\Theta}))^2 \tag{3}$$

is a very popular loss function for time series prediction; not rarely, a regularisation penalty term is introduced to guide the minimisation (learning) procedure towards solution incorporating some wished property, for example, predictor smoothness as well as mitigating occurrence of model overfitting.

$$\mathcal{L}(\mathbf{\Theta}) = \frac{1}{N} \sum_{t=1}^{N} (\mathbf{y_t} - f(\mathbf{x_t}; \mathbf{\Theta}))^2 + \mathbf{\Omega}(\mathbf{\Theta}). \tag{4}$$

The most used regularisation scheme controlling model complexity is the L2 regularisation $\mathbf{\Omega}(\mathbf{\Theta}) = \lambda \|\mathbf{\Theta}\|_2^2$, being λ a suitable hyperparameter controlling the intensity of the regularisation.

As Equation (4) is not convex, the solution cannot be obtained in a closed form with linear equation solvers or convex optimisation techniques. Parameter estimation (learning procedure) operates iteratively for example, by leveraging on the gradient descent approach:

$$\mathbf{\Theta}_k = \mathbf{\Theta}_{k-1} - \eta \nabla_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta})\,|_{\mathbf{\Theta}=\mathbf{\Theta}_{k-1}} \tag{5}$$

where η is the learning rate and $\nabla_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta})$ the loss function gradient with respect to $\mathbf{\Theta}$. Stochastic gradient descent, RMSProp [44], Adagrad [45], Adam [46] are popular learning procedures. The learning procedure yields estimate $\hat{\mathbf{\Theta}} = \mathbf{\Theta}_k$ associated with the predictive model $f(\mathbf{x_t}; \hat{\mathbf{\Theta}})$.

Here, we consider deep FNNs to be the baseline architectures.

In multi-step-ahead prediction, the output layer dimension coincides with the forecasting horizon $n_O > 1$. The dimension of the input vector depends also on the presence of exogenous variables; this aspect is further discussed in Section 7.

3.1 | FNNs' application for short-term load forecasting

The use of feed-forward neural networks in short-term load forecasting dates back to the 90s. Authors in [11] propose a shallow neural network with a single hidden layer to provide a 24-h forecast using both load and temperature information. In [10], one-day-ahead forecast is implemented using two different prediction strategies: one network provides all 24 forecast values in a single shot (MIMO strategy) while another single output network provides the day-ahead prediction by recursively feedbacking its last value estimate (recurrent strategy). The recurrent strategy proves to be more efficient in terms of both training time and forecasting accuracy. In [47], the authors present a feed-forward neural network to forecast electric loads on a weekly basis. The sparsely connected feed-forward architecture receives the load time series, temperature readings, as well as the (coded) time and day of the week. It is shown that the extra information improves the forecast accuracy compared to an ARIMA model trained on the same task. [12] presents one of the first multi-layer FNNs to forecast the hourly load of a power system.

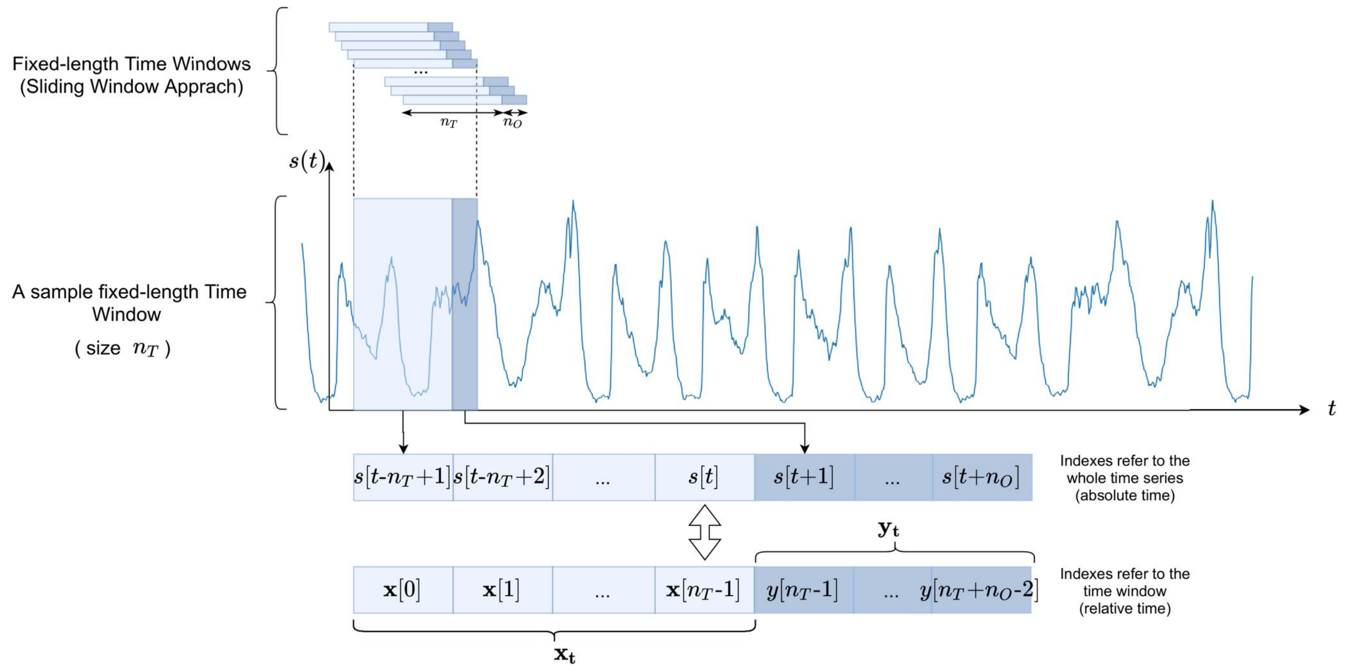
Fixed-length Time Windows (Sliding Window Apprach)
n_T
n_O
$s(t)$
A sample fixed-length Time Window
(size n_T)
t
$s[t\text{-}n_T\text{+}1]$
$s[t\text{-}n_T\text{+}2]$
...
$s[t]$
$s[t\text{+}1]$
...
$s[t\text{+}n_O]$
Indexes refer to the whole time series (absolute time)
$\mathbf{y_t}$
$\mathbf{x}[0]$
$\mathbf{x}[1]$
...
$\mathbf{x}[n_T\text{-}1]$
$y[n_T\text{-}1]$
...
$y[n_T\text{+}n_O\text{-}2]$
Indexes refer to the time window (relative time)
$\mathbf{x_t}$

FIGURE 1 A sliding windowed approach is used to frame the forecasting problem into a supervised machine learning problem. The target signal **s** is split in multiple input output pairs $(\mathbf{x_t}, \mathbf{y_t})\ \forall\ t \in \{n_T, n_{T+1}, \ldots, T - n_0\}$

A detailed review concerning applications of artificial neural networks in short-term load forecasting can be found in [3]. However, this survey dates back to the early 2000s and does not discuss deep models. More recently, architectural variants of feed-forward neural networks have been used; for example, in [14], a ResNet [48] inspired model is used to provide day-ahead forecast by leveraging on a very deep architecture. The article shows a significant improvement on aggregated load forecasting when compared to other (not neural) regression models on different datasets.

4 | RECURRENT NEURAL NETWORKS

In this section, we overview recurrent neural networks and, in particular, the Elmann Net architecture [49], long short-term memory [50] and the gated recurrent Unit [51] networks. Afterwards, we introduce deep recurrent neural networks and discuss different strategies to perform multi-step-ahead forecasting. Finally, we present related work in short-term load forecasting that leverages on recurrent networks.

4.1 | Elmann RNNs

Elmann recurrent neural networks (ERNN) were proposed in [49] to generalise feed-forward neural networks for better handling ordered data sequences like time series.

The reason behind the effectiveness of RNNs in dealing with sequences of data is their ability to learn a compact representation of the input sequence $\mathbf{x_t}$ by means of a recurrent function f that implements mapping:

$$\mathbf{h}[t] = f(\mathbf{h}[t-1], \mathbf{x}[t]; \mathbf{\Theta}) \tag{6}$$

The architecture is depicted in Figure 2.

By expanding Equation (6) and given sequence $\mathbf{x_t} = [\mathbf{x}[0], \ldots, \mathbf{x}[n_T - 1]]$, $\mathbf{x}[t] \in \mathrm{IR}^d$ the neural computation satisfies the following equation:

$$\mathbf{a}[t] = \mathbf{W}^T\mathbf{h}[t-1] + \mathbf{U}^T\mathbf{x}[t] \tag{7}$$

$$\mathbf{h}[t] = \phi(\mathbf{a}[t]) \tag{8}$$

$$\mathbf{y}[t] = \psi(\mathbf{V}^T\mathbf{h}[t]) \tag{9}$$

where $\mathbf{W} \in \mathrm{IR}^{n_H \times n_H}$, $\mathbf{U} \in \mathrm{IR}^{d \times n_H}$, $\mathbf{V} \in \mathrm{IR}^{n_H \times n_O}$ are the weight matrices for hidden-hidden, input-hidden, and hidden-output connections, respectively, $\phi(\cdot)$ is an activation function (generally the hyperbolic tangent one) and $\psi(\cdot)$ is normally a linear function. The computation of a single module in an Elmann recurrent neural network is depicted in Figure 3.

It can be noted that an ERNN processes one element of the sequence at a time, preserving its inherent temporal order. After reading an element from the input sequence $\mathbf{x}[t] \in \mathrm{IR}^d$, the network updates its internal state $\mathbf{h}[t] \in \mathrm{IR}^{n_H}$ using both (a transformation of) the latest state $\mathbf{h}[t-1]$ and (a transformation of) the current input (Equation 6). The described process can be better visualised as an acyclic graph obtained from the original cyclic graph (left side of Figure 2) via an operation known as time unfolding (right side of Figure 2). It is of fundamental importance to point out that all nodes in the unfolded network share the same parameters, as they are just replicas distributed over time.

TABLE 2 The nomenclature used in this work

Notation	Description
n_T	Window size of the regressor vector
n_O	Time horizon of the forecast
$d-1$	Number of exogenous variable
u	Scalar value
$\mathbf{u}$	Vector/matrix
$\mathbf{u}^{\mathrm{T}}$	Vector/matrix transposed
$\odot$	Elementwise product
$*$	Convolution operator
$*_d$	Dilated convolution operation
ℓ	Index of the l^{th} layer
$\mathbf{x_t}$	Regressor vector at discrete time t (reference system: time series) $\mathbf{x_t} \in \mathrm{IR}^{n_T}$ or $\mathbf{x_t} \in \mathrm{IR}^{n_T \times d}$
$\mathbf{y_t}$	True output for the input sequence at time t (reference system: time series) $\mathbf{y_t} \in \mathrm{IR}^{n_O}$
$\hat{\mathbf{y}}_\mathbf{t}$	Predicted output for the input sequence at time t (reference system: time series) $\hat{\mathbf{y}}_\mathbf{t} \in \mathrm{IR}^{n_O}$
$\mathbf{x}[t]$	Input vector of load and other features at time t (reference system: time window) $\mathbf{x}[t] \in \mathrm{IR}$ or $\mathbf{x}[t] \in \mathrm{IR}^d$
$y[t]$	Value of the load time series at time $t+1$ (reference system: time window) $y[t] \in \mathrm{IR}$
$\mathbf{z_t}$	Exogenous features' vector at time t (reference system: time series) $\mathbf{z_t} \in \mathrm{IR}^{n_T \times d-1}$
$\mathbf{z}[t]$	Exogenous features' vector at time t (reference system: time window). $\mathbf{z}[t] \in \mathrm{IR}^{d-1}$
$\mathbf{h}$	Hidden state vector
$\mathbf{\Theta}$	Model's vector of parameters
n_H	Number of hidden neurons

The parameters of the network $\mathbf{\Theta} = [\mathbf{W}, \mathbf{U}, \mathbf{V}]$ are usually learnt via backpropagation through time (BPTT) [52, 53], a generalised version of standard backpropagation. In order to apply gradient-based optimisation, the recurrent neural network has to be transformed through the unfolding procedure shown in Figure 2. In this way, the network is converted into a FNN having as many layers as time intervals in the input sequence, and each layer is constrained to have the same weight matrices. In practice, truncated backpropagation through time [54] (TBPTT) (τ_b, τ_f) is used. The method processes an input window of length n_T one timestep at a time and runs BPTT for τ_b timesteps every τ_f steps. Notice that having $\tau_b < n_T$ does not limit the memory capacity of the network as the hidden state incorporates information taken from the whole sequence. Despite that, setting τ_b to a very low number may result in poor performance. In the literature, BPTT is considered equivalent to TBPTT $(\tau_b = n_T,\ \tau_f = 1)$. In this work, we used epoch-wise truncated BPTT that is, TBPTT

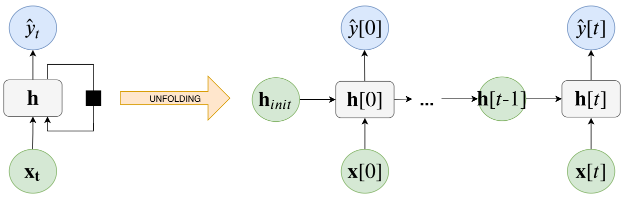

FIGURE 2 (Left) A simple RNN with a single input. The black box represents the delay operator which leads to Equation (6). (Right) The network after unfolding. Note that the structure reminds that of a (deep) feedforward neural network but, here, each layer is constrained to share the same weights. $\mathbf{h}_{\text{init}}$ is the initial state of the network which is usually set to zero

$(\tau_b = n_T,\ \tau_f = n_T)$ to indicate that the weights' update is performed once a whole sequence has been processed.

Despite the model simplicity, Elmann RNNs are hard to train due to the ineffectiveness of gradient (back)propagation. In fact, it emerges that the propagation of the gradient is effective for short-term connections but is very likely to fail for long-term ones, when the gradient norm usually shrinks to zero or diverges. These two behaviours are known as the vanishing gradient and the exploding gradient problems [55, 56] and are extensively studied in the machine learning community.

4.2 | Long short-term memory

Recurrent neural networks with long short-term memory (LSTM) were introduced to cope with the vanishing and exploding gradients' problems occurring in ERNNs and, more in general, in standard RNNs [50]. LSTM networks maintain the same topological structure of ERNN but differ in the composition of the inner module—or cell.

Each LSTM cell has the same input and output as an ordinary ERNN cell but, internally, it implements a gated system that controls the neural information processing (see Figures 3 and 4). The key feature of gated networks is their ability to control the gradient flow by acting on the gate values; this allows the learning procedure to tackle the vanishing gradient problem, as LSTM can maintain its internal memory unaltered for long time intervals. Notice from the equations below that the inner state of the network results as a linear combination of the old state and the new state (Equation 14). Part of the old state is preserved and flows forward while in the ERNN, the state value is completely replaced at each timestep (Equation 8). In detail, the neural computation is

$$\mathbf{i}[\mathbf{t}] = \psi(\mathbf{W_f}\mathbf{h}[t-1] + \mathbf{U_f}\mathbf{x}[t]) \tag{10}$$

$$\mathbf{f}[\mathbf{t}] = \psi(\mathbf{W_i}\mathbf{h}[t-1] + \mathbf{U_i}\mathbf{x}[t]) \tag{11}$$

$$\mathbf{o}[\mathbf{t}] = \psi(\mathbf{W_o}\mathbf{h}[t-1] + \mathbf{U_o}\mathbf{x}[t]) \tag{12}$$

$$\widetilde{\mathbf{c}}[\mathbf{t}] = \phi(\mathbf{W_c}\mathbf{h}[t-1] + \mathbf{U_c}\mathbf{x}[t]) \tag{13}$$

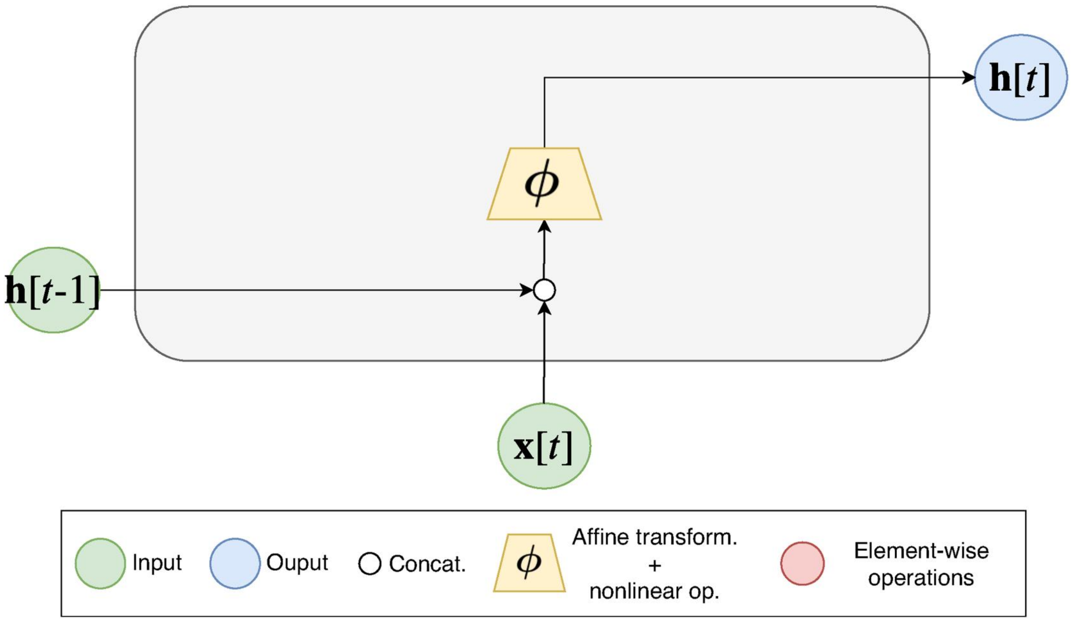

FIGURE 3 A simple Elmann recurrent neural network block with one cell implementing Equation (7) and (8) once rewritten as matrix concatenation: $\mathbf{a}[t] = [\mathbf{W}, \mathbf{U}]^T[\mathbf{h}[t-1], \mathbf{x}[t]], \mathbf{h}[t] = \varphi(\mathbf{a}[t])$, with $[\mathbf{W}, \mathbf{U}] \in \mathrm{IR}^{(n_H+d)\times n_H}$ and $[\mathbf{h}[t-1], \mathbf{x}[t]] \in \mathrm{IR}^{n_H+d}$, Usually $\varphi(\cdot)$ is the hyperbolic tangent

$$\mathbf{c}[\mathbf{t}] = \mathbf{f}[t] \odot \mathbf{c}[t-1] + \mathbf{i}[t] \odot \widetilde{\mathbf{c}}[t] \tag{14}$$

$$\mathbf{h}[\mathbf{t}] = \mathbf{o}[t] \odot \phi(\mathbf{c}[t]) \tag{15}$$

where $\mathbf{W_f}, \mathbf{W_i}, \mathbf{W_o}, \mathbf{W_c} \in \mathrm{IR}^{n_H \times n_H}$, $\mathbf{U_f}, \mathbf{U_i}, \mathbf{U_o}, \mathbf{U_c} \in \mathrm{IR}^{n_H \times d}$ are parameters to be learnt, $\odot$ is the Hadamard product, $\psi(\cdot)$ is generally a sigmoid activation while $\phi(\cdot)$ can be any non-linear one (hyperbolic tangent in the original paper). The cell state $\mathbf{c}[t]$ encodes the—so far learnt—information from the input sequence. At timestep t, the flow of information within the unit is controlled by three elements called gates: the forget gate $\mathbf{f}[t]$ controls the cell state's content and changes it when obsolete, the input gate $\mathbf{i}[t]$ controls which state value and how much will be updated, $\widetilde{\mathbf{c}}[t]$, finally the output gate $\mathbf{o}[t]$ produces a filtered version of the cell state and serves it as the network's output $\mathbf{h}[t]$ [57].

4.3 | Gated recurrent units

First introduced in [51], gated recurrent units (GRUs) are a simplified variant of LSTM and, as such, belong to the family of gated RNNs. GRUs distinguish themselves from LSTMs for merging in one-gate functionalities controlled by the forget gate and the input gate. This kind of cell ends up having just two gates, which results in a more parsimonious architecture compared to LSTM that, instead, has three gates.

The basic components of a GRU cell are outlined in Figure 5, whereas the neural computation is controlled by:

$$\mathbf{u}[\mathbf{t}] = \psi(\mathbf{W_u}\mathbf{h}[t-1] + \mathbf{U_u}\mathbf{x}[t]) \tag{16}$$

$$\mathbf{r}[\mathbf{t}] = \psi(\mathbf{W_r}\mathbf{h}[t-1] + \mathbf{U_r}\mathbf{x}[t]) \tag{17}$$

$$\widetilde{\mathbf{h}}[\mathbf{t}] = \phi(\mathbf{W_c}(\mathbf{r}[t] \odot \mathbf{h}[t-1]) + \mathbf{U_c}\mathbf{x}[t]) \tag{18}$$

$$\mathbf{h}[\mathbf{t}] = \mathbf{u}[\mathbf{t}] \odot \mathbf{h}[t-1] + (1 - \mathbf{u}[t]) \odot \widetilde{\mathbf{h}}[t] \tag{19}$$

where $\mathbf{W_u}, \mathbf{W_r}, \mathbf{W_c} \in \mathrm{IR}^{n_H \times n_H}$, $\mathbf{U_u}, \mathbf{U_r}, \mathbf{U_c} \in \mathrm{IR}^{n_H \times d}$ are the parameters to be learnt, $\psi(\cdot)$ is generally a sigmoid activation, while $\phi(\cdot)$ can be any kind of non-linearity (in the original work it was a hyperbolic tangent). $\mathbf{u}[t]$ and $\mathbf{r}[t]$ are the update and reset gates, respectively. Several works in the natural language processing community show that GRUs perform comparably to LSTM but generally train faster due to their lighter computation [58, 59].

4.4 | Deep recurrent neural networks

All recurrent architectures presented so far are characterised by a single layer. In turn, this implies that the computation is composed of an affine transformation followed by non-linearity. That said, the concept of depth in RNN is less straightforward than in feedforward architectures. Indeed, the latter ones become deep when the input is processed by a large number of non-linear transformations before generating the output values. However, according to this definition, an unfolded RNN is already a deep model given its multiple non-linear processing layers. That said, a deep multi-level processing can be applied to all the transition functions (input-hidden, hidden-hidden, and hidden-output) as there are no intermediate layers involved in these computations [60]. Deepness can also be introduced in recurrent neural networks by stacking

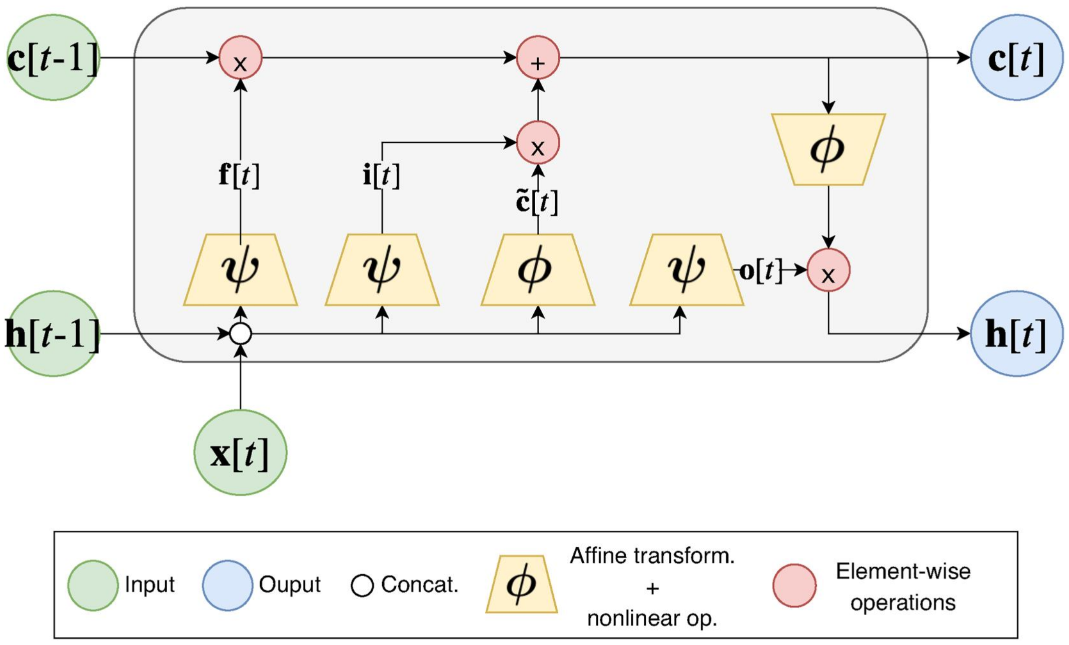

FIGURE 4 Long-Short Term Memory block with one cell

recurrent layers one on top of the other [61]. As this deep architecture is more intriguing, in this work, we refer to it as a Deep RNN. By iterating the RNN computation, the function implemented by the deep architecture can be represented as

$$\mathbf{h}_\ell[t] = f(\mathbf{h}_\ell[t-1], \mathbf{h}_{\ell-1}[t]; \boldsymbol{\Theta}) \qquad \ell = 1, 2, \dots, L \tag{20}$$

where $\mathbf{h}_\ell[t]$ is the hidden state at timestep t for layer ℓ. Notice that $\mathbf{h}_0[t] = \mathbf{x}[t]$. It has been empirically shown that Deep RNNs better capture the temporal hierarchy exhibited by time series then their shallow counterpart [60, 62, 63]. Of course, hybrid architectures having different layers—recurrent or not—can be considered as well.

4.5 | Multi-step prediction schemes

There are five different architecture-independent strategies for multi-step-ahead forecasting [64]:

4.5.1 | Recursive strategy (Rec)

A single model is trained to perform a one-step-ahead forecast given the input sequence. Subsequently, during the operational phase, the forecasted output is recursively fed back and considered to be the correct one. By iterating this procedure n_O times, we generate the forecast values at time $t + n_O$. The procedure is described in Algorithm 1, where $\mathbf{x}[1:]$ is the input vector without its first element while the *vectorize*$(\cdot)$ procedure concatenates the scalar output y to the exogenous input variables.

Algorithm 1 Recursive Strategy (Rec) for Multi-Step Forecasting

```
1: $\mathbf{x} \leftarrow \mathbf{x}_t$
2: $\mathbf{o} \leftarrow$ empty list
3: $k \leftarrow 1$
4: while $k < n_O + 1$ do
5:   $o \leftarrow f(\mathbf{x})$
6:   $\mathbf{o} \leftarrow concatenate(\mathbf{o}, o)$
7:   $\mathbf{x} \leftarrow concatenate(\mathbf{x}[1:], vectorize(o))$
8:   $k \leftarrow k + 1$
9: end while
10: return $\mathbf{o}$ as $\hat{\mathbf{y}}_t$
```

To summarise, the predictor f receives as input a vector $\mathbf{x}$ of length n_T and outputs a scalar value o.

4.5.2 | Direct strategy

Design a set of n_O independent predictors $f_k, k = 1, \dots, n_O$, each of which provides a forecast at time $t + k$. Similar to the recursive strategy, each predictor f_k outputs a scalar value o, but the input vector is the same for all the predictors. Algorithm 2 details the procedure.

Algorithm 2 Direct Strategy for Multi-step Forecasting

```
1: $\mathbf{x} \leftarrow \mathbf{x}_t$
2: $\mathbf{o} \leftarrow$ empty list
3: $k \leftarrow 1$
4: while $k < n_O + 1$ do
```

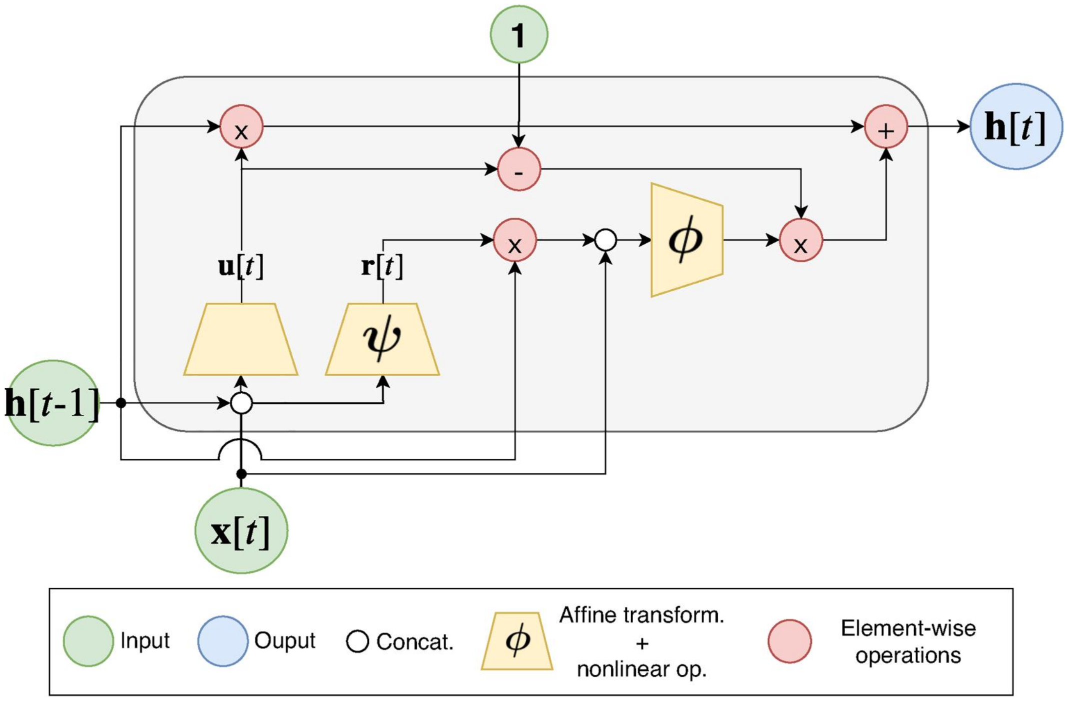

FIGURE 5 Gated Recurrent Unit memory block with one cell

```
5: $\mathbf{o} \leftarrow concatenate(\mathbf{o}, f_k(\mathbf{x}))$
6:   $k \leftarrow k + 1$
7: end while
8: return $\mathbf{o}$ as $\hat{\mathbf{y}}_t$
```

4.5.3 | DirRec strategy

[65] is a combination of the above two strategies. Similar to the direct approach, n_O models are used, but here, each predictor leverages on an enlarged input set, obtained by adding the results of the forecast at the previous timestep. The procedure is detailed in Algorithm 3.

Algorithm 3 Direct Strategy for Multi-step Forecasting

```
1: $\mathbf{x} \leftarrow \mathbf{x}_t$
2: $\mathbf{o} \leftarrow$ empty list
3: $k \leftarrow 1$
4: while $k < n_O + 1$ do
5: $o \leftarrow f_k(\mathbf{x})$
6:   $\mathbf{o} \leftarrow concatenate(\mathbf{o}, o)$
7:   $\mathbf{x} \leftarrow concatenate(\mathbf{x}, vectorize(o))$
8:   $k \leftarrow k + 1$
9: end while
10: return $\mathbf{o}$ as $\hat{\mathbf{y}}_t$
```

4.5.4 | MIMO strategy

Multiple input-Multiple output [66], a single predictor f is trained to forecast a whole output sequence of length n_O in one shot, that is, different from the previous cases, the output of the model is not a scalar but a vector

$$\hat{\mathbf{y}}_t = f(\mathbf{x}_t)$$

4.5.5 | DIRMO strategy

[67], represents a trade-off between the direct strategy and the MIMO strategy. It divides the n_O steps forecasts into smaller forecasting problems, each of which is of length s. It follows that $\lceil \frac{n_O}{s} \rceil$ predictors are used to solve the problem.

Given the considerable computational demand required by RNNs during training, we focus on multi-step forecasting strategies that are computationally cheaper, specifically, recursive and MIMO strategies [64]. We will call them RNN-Rec and RNN-MIMO.

Given the hidden state $\mathbf{h}[t]$ at timestep t, the hidden-output mapping is obtained through a fully connected layer on top of the recurrent neural network. The objective of this dense network is to learn the mapping between the last state of the recurrent network, which represents a kind of lossy summary of the task-relevant aspect of the input sequence and the output domain. This holds for all the presented recurrent networks and is consistent with Equation (9). In this work, RNN-Rec and RNN-MIMO differ in the cardinality of the output domain, which is 1 for the former and n_O for the latter, meaning that in Equation (9) either $\mathbf{V} \in \mathrm{IR}^{n_H \times 1}$ or $\mathbf{V} \in \mathrm{IR}^{n_H \times n_O}$. The objective function is

$$\mathcal{L}(\mathbf{\Theta}) = \frac{1}{n_O} \sum_{t=0}^{n_O - 1} (y[t] - \hat{y}[t])^2 + \mathbf{\Omega}(\mathbf{\Theta}) \qquad (21)$$

4.6 | RNNs' application for short-term load forecasting

In [17], an Elmann recurrent neural network is considered to provide hourly load forecasts. The study also compares the performance of the network when additional weather information such as temperature and humidity are fed to the model. The authors conclude that, as expected, the recurrent network benefits from multi-input data and, in particular, weather ones. [28] makes use of ERNN to forecast household electric consumption obtained from a suburban area in the neighbouring areas of Palermo (Italy). In addition to the historical load measurements, the authors introduce several features to enhance the model's predictive capabilities. Besides the weather and the calendar information, a specific ad hoc index was created to assess the influence of the use of air conditioning equipment on the electricity demand. In recent years, LSTMs have been adopted in short-term load forecasting, proving to be more effective than traditional time series analysis methods. In [21], LSTM is shown to outperform traditional forecasting methods as it is able to exploit the long-term dependencies in the time series to forecast the day-ahead load consumption. Several studies proved to be successful in enhancing the recurrent neural network capabilities by employing multivariate input data. In [22], the authors propose a deep, LSTM-based architecture that uses past measurements of the whole household consumption along with some measurements from selected appliances to forecast the consumption of the subsequent time interval (i.e., a one-step prediction). In [23], a LSTM-based network is trained using a multivariate input, which includes temperature, holiday/working day information, and date and time information. Similarly, in [31], a power demand forecasting model based on LSTM shows an accuracy improvement compared to more traditional machine learning techniques such as gradient boosting trees and support vector regression.

GRUs have not been used much in the literature as LSTM networks are often preferred. That said, the use of GRU-based networks is reported in [18], while a more recent study [24] uses GRUs for the daily consumption forecast of individual customers. Thus, investigating deep GRU-based architectures is a relevant scientific topic, also thanks to their faster convergence and simpler structure compared to LSTM [58].

Despite all these promising results, an extensive study of recurrent neural networks [18], and in particular of ERNN, LSTM, GRU, ESN [19] and NARX, concludes that none of the investigated recurrent architectures manages to outperform the others in all considered experiments. Moreover, the authors noticed that recurrent cells with gated mechanisms like LSTM and GRU perform comparably well than much simpler ERNN. This may indicate that in short-term load forecasting, the gating mechanism may be unnecessary; this issue is further investigated—and evidence found—in the present work.

As a final comment, we observe that good results have been achieved in the past by using diversified computing architectures and hand-crafted features extracted from available data streams. Readers might then perceive this forecasting problem as solved and the exercise perform here purely academic. This is not true in several ways because (1) different computing architectures expose different approximation abilities. As such, for a given problem, one should test different non-linear families and identify the optimal one. (2) Huge data availability allows us to consider more complex—deep—architectures that can even tackle both fast and slow dynamics within the same model. The approximation error, i.e., the discrepancy between the unknown dynamics describing the power consumption and those provided by the approximating one reduces. (3) From (2), it should be noted that even a very small (statistically sound) improvement in forecasting accuracy has a huge monetary and market impact—not to mention the sustainability asset. (4) Moreover, many dated papers suffer from inaccuracies in presenting the forecast performance as statistically sound assessments (e.g., K-fold cross-validation and statistical tests) have not been considered. (5) Finally, some deep learning architectures naturally learn in the first computational layers the best features solving the problem; this is an extra value that does not require any more handcraft feature design. That said, valuable handcraft features can always be taken into account and integrated with automatically extracted ones to further favour the inference performance.

5 | SEQUENCE-TO-SEQUENCE MODELS

Sequence-to-sequence (seq2seq) architectures [68] or encoder–decoder models [51] were initially designed to solve RNNs' inability to produce output sequences of arbitrary length. The architecture was first used in neural machine translation [51, 69, 70] but has emerged as the golden standard in different fields such as speech recognition [62, 71, 72] and image captioning [73].

The core idea of this general framework is to employ two networks resulting in an encoder–decoder architecture. The first neural network (possibly deep) f—the encoder—reads the input sequence $\mathbf{x_t} \in \mathrm{IR}^{n_T \times d}$ of length n_T one timestep at a time; the computation generates a, generally lossy, fixed dimensional vector representation of it $\mathbf{c} = f(\mathbf{x_t}, \mathbf{\Theta}_f)$, $\mathbf{c} \in \mathrm{IR}^{d'}$. This embedded representation is named context and can be the last hidden state of the encoder or a function of it. Then, a second neural network g—the decoder—learns how to produce the output sequence $\mathbf{\hat{y}_t} \in \mathrm{IR}^{n_O}$ given the context vector, that is, $\mathbf{\hat{y}} = g(\mathbf{c}, \mathbf{\Theta}_g)$. The schematics of the whole architecture is depicted in Figure 6.

The encoder and the decoder modules are generally two recurrent neural networks trained together to minimise the objective function:

$$\mathcal{L}(\mathbf{\Theta}) = \sum_{t=0}^{n_O-1} (y[t] - \hat{y}[t])^2 + \mathbf{\Omega}(\mathbf{\Theta}), \quad \mathbf{\Theta} = [\mathbf{\Theta}_f, \mathbf{\Theta}_g] \tag{22}$$

$$\hat{y}[t] = g(y[t-1], \mathbf{h}[t-1], \mathbf{c}; \mathbf{\Theta}) \tag{23}$$

where $\hat{y}[t]$ is the decoder's estimate at time t, $y[t]$ is the real measurement, $\mathbf{h}[t-1]$ is the decoder's last state, $\mathbf{c}$ is the context vector from the encoder, $\mathbf{x}$ is the input sequence and $\mathbf{\Omega}(\mathbf{\Theta})$ the regularisation term. The training procedure for this type of architecture is called teacher forcing [74]. As shown in Figure 6 and explained in Equation (23), during training, the decoder's input at time t is the ground-truth value $y[t-1]$, which is then used to generate the next state $\mathbf{h}[t]$ and, then, the estimate $\hat{y}[t]$. During inference, the true values are unavailable and replaced by the estimates:

$$\hat{y}[t] = g(\hat{y}[t-1], \mathbf{h}[t-1], \mathbf{c}; \mathbf{\Theta}). \tag{24}$$

This discrepancy between training and testing results in errors accumulating over time during inference. In the literature, this problem is often referred to as exposure bias [75]. Several solutions have been proposed to address this problem; in [76], the authors present scheduled sampling, a curriculum learning strategy that gradually changes the training process by switching the decoder's inputs from ground-truth values to the model's predictions. The *professor forcing* algorithm, introduced in [77], uses an adversarial framework to encourage the dynamics of the recurrent network to be the same both at training and operational (test) time. Finally, in recent years, reinforcement learning methods have been adopted to train sequence-to-sequence models; a comprehensive review is presented in [78].

In this work, we investigate two sequence-to-sequence architectures, one trained via *teacher forcing* (TF) and one using *self-generated* (SG) samples. The former is characterised by Equation (23) during training while Equation (24) is used during prediction. The latter architecture adopts Equation (24) both for training and prediction. The decoder's dynamics are summarised in Figure 7. It is clear that the two training procedures differ in the decoder's input source: ground-truth values in teacher forcing, estimated values in self-generated training.

5.1 | seq2seq application for short-term load forecasting

Only recently, seq2seq models have been adopted in short-term load forecasting. In [33], a LSTM-based encoder–decoder model is shown to produce superior performance compared to the standard LSTM. In [79], the authors introduce an adaptation of RNN-based sequence-to-sequence architectures for time series forecasting of electrical loads to demonstrate its better performance with respect to a suite of models ranging from standard RNNs to classical time series techniques. The authors in [37] provide a similar empirical proof of the superior performance provided by sequence-to-sequence model when compared to standard deep neural network. Moreover, the study also suggests that using LSTM or GRU cells within the seq2seq architecture shell provides better results when compared against the Elmann unit. More recent studies in the field also consider an additional attention mechanism to the sequence-to-sequence model [38], which provides a more effective modelling of long-term temporal sequences while not requiring incredibly long input. The attention mechanism has been shown to provide good results in multi-step-ahead load forecasting both with univariate and multivariate time series [36].

6 | CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) [80] are a family of neural networks designed to work with data that can be structured in a grid-like topology. CNNs were originally used on two-dimensional and three-dimensional images, but they are also suitable for one-dimensional data such as univariate time series. Once recognised as a very efficient solution for image recognition and classification [48, 81–83], CNNs have experienced wide adoption in many different computer vision tasks [84–88]. Moreover, sequence modelling tasks, such as short-term electric load forecasting, have been mainly addressed with recurrent neural networks, but recent research indicates that convolutional networks can also attain state-of-the-art performance in several applications including audio generation [89], machine translation [90] and time series prediction [91].

As the name suggests, these kind of networks are based on a discrete convolution operator that produces an output feature map $\mathbf{f}$ by sliding a kernel $\mathbf{w}$ over the input $\mathbf{x}$.

Each element in the output feature map is obtained by summing up the result of the element-wise multiplication between the input patch (i.e., a slice of the input having the same dimensionality as the kernel) and the kernel. The number of kernels (filters) M used in a convolutional layer determines the depth of the output volume (i.e., the number of output feature maps). To control the other spatial dimensions of the output feature maps, two hyperparameters are used: stride and padding. Stride represents the distance between two consecutive input patches and can be defined for each direction of

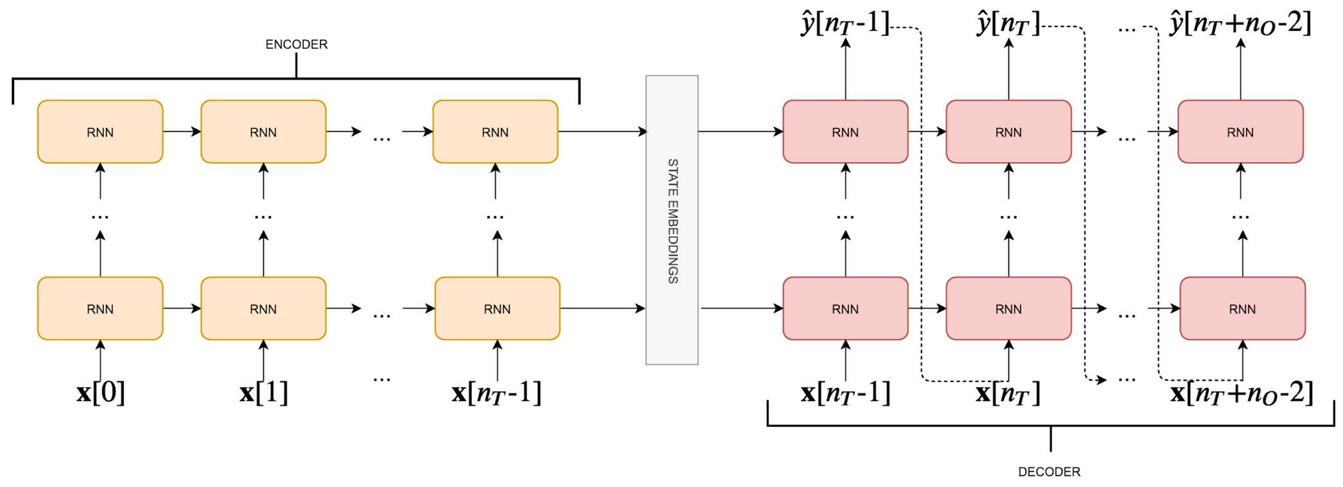

FIGURE 6 seq2seq (Encoder-Decoder) architecture with a general Recurrent Neural network both for the encoder and the decoder networks. Assuming a Teacher Forcing training process, the solid lines in the decoder represent the training phase while the dotted lines depict the values path during prediction

motion. Padding refers to the possibility of implicitly enlarging the inputs by adding (usually) zeros at the borders to control the output size. Indeed, without padding, the dimensionality of the output would be reduced after each convolutional layer.

Considering a 1-D time series $\mathbf{x} \in \mathrm{IR}^{n_T}$ and a one-dimensional kernel $\mathbf{w} \in \mathrm{IR}^{k}$, the i^{th} element of the convolution between $\mathbf{x}$ and $\mathbf{w}$ is

$$f(i) = (\mathbf{x} * \mathbf{w})(i) = \sum_{j=0}^{k-1} x(i-j)w(j) \tag{25}$$

with $\mathbf{f} \in \mathrm{IR}^{n_T-k+1}$ if no zero-padding is used, otherwise the padding matches the input dimensionality, that is, $\mathbf{f} \in \mathrm{IR}^{n_T}$. Equation (25) is referred to as the one-dimensional input case but can be easily extended to multi-dimensional inputs (e.g., images, where $\mathbf{x} \in \mathrm{IR}^{W \times H \times D}$) [92]. The reason behind the success of these networks can be summarised in the following four points:

- Local connectivity: each hidden neuron is connected to a subset of input neurons that are close to each other (according to a specific spatio–temporal metric). This property allows the network to drastically reduce the number of parameters to learn (with respect to a fully connected network) and facilitate computations.
- Parameter sharing: the weights used to compute the output neurons in a feature map are the same so that the same kernel is used for each location. This allows to reduce the number of parameters to learn.
- Translation equivariance: the network is robust to an eventual shifting of its input.
- Features free: these architectures free the designer from selecting and extracting handcraft features—a hard problem per se—as they are automatically extracted by the first learnt processing layers of the CNN.

In our work, we focus on a convolutional architecture inspired by Wavenet [89], a fully probabilistic and autoregressive model used for generating raw audio waveforms and extended to time series prediction tasks [91].

To the best of the authors' knowledge, this architecture has never been proposed to forecast electric load. A recent empirical comparison between temporal convolutional networks and recurrent networks has been carried out in [93] on tasks such as polymorphic music and charter-sequence-level modelling. The authors were the first to use the name temporal convolutional networks (TCNs) to indicate convolutional networks that are autoregressive, able to process sequences of arbitrary length and output a sequence of the same length. To achieve the above, the network has to employ causal (dilated) convolutions, and residual connections should be used to handle a very long history size.

6.1 | Dilated causal convolution

Being TCNs, a family of autoregressive models, the estimated value at time t must depend only on past samples and not on future ones (Figure 8). To achieve this behaviour in a convolutional neural network, the standard convolution operator is replaced by causal convolution. Moreover, zero-padding of length (filter size – 1) is added to ensure that each layer has the same length of the input layer. To further enhance the network capabilities, *dilated causal convolutions* (DCCs) are used, allowing to increase the receptive field of the network (i.e., the number of input neurons to which the filter is applied) and its ability to learn long-term dependencies in the time series (see Figure 9). Given a one-dimensional input $\mathbf{x} \in \mathrm{IR}^{n_T}$, and a kernel $\mathbf{w} \in \mathrm{IR}^{k}$, a dilated convolution output using a dilation factor d becomes

$$f(i) = (\mathbf{x} *_d \mathbf{w})(i) = \sum_{j=0}^{k-1} x(i-dj)w(j) \tag{26}$$

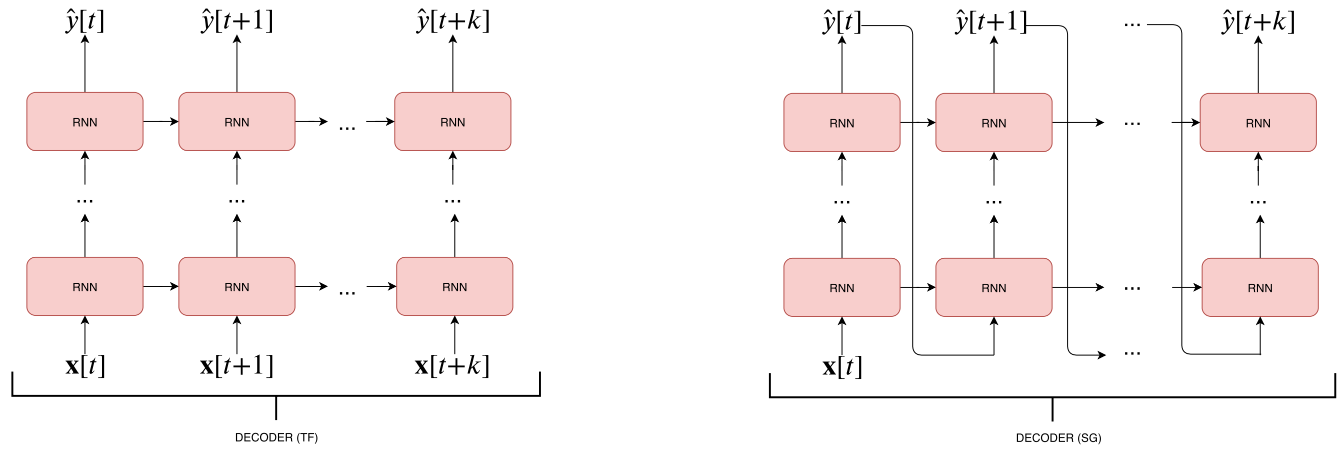

FIGURE 7 (Left) decoder with ground-truth inputs (Teacher Forcing). (Right) Decoder with self-generated inputs

This is a major advantage with respect to simple causal convolutions, as in the latter case the receptive field r grows linearly with the depth of the network $r = k(L-1)$, while with dilated convolutions the dependence is exponential $r = 2^{L-1}k$, ensuring that a much larger history size is used by the network.

6.2 | Residual connections

Despite the implementation of dilated convolution, the CNN still needs a large number of layers to learn the dynamics of the inputs. Moreover, the performance often degrades with the increase of the network depth. The degradation problem has been first addressed in [48], where the authors propose a deep residual learning framework. The authors observe that for a L-layers network with a training error ϵ, inserting k extra layers on top of it should either leave the error unchanged or improve it. Indeed, in the worst case scenario, the new k stacked non-linear layers should learn the identity mapping $\mathbf{y} = \mathcal{H}(\mathbf{x}) = \mathbf{x}$, where $\mathbf{x}$ is the output of the network having L layers and $\mathbf{y}$ is the output of the network with $L+k$ layers. Although almost trivial, in practice, neural networks experience problems in learning this identity mapping. The proposed solution suggests that these stacked layers fit a residual mapping $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ instead of the desired one, $\mathcal{H}(\mathbf{x})$. The original mapping is recast into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$, which is realised by feed-forward neural networks with shortcut connections; in this way, the identity mapping is learnt by simply driving the weights of the stacked layers to zero.

By means of the two aforementioned principles, the temporal convolutional network is able to exploit a large history size in an efficient manner. Indeed, as observed in [93], these models present several computational advantages compared to RNNs. In fact, they have lower memory requirements during training and the predictions for later timesteps are not done sequentially but can be computed in parallel exploiting parameter sharing. Moreover, TCNs' training is much more stable than that involving RNNs, allowing to avoid the exploding/vanishing gradient problem. For all of the above, TCNs have demonstrated to be a promising area of research for time series prediction problems, and here, we aim to assess their forecasting performance with respect to state-of-the-art models in short-term load forecasting. The architecture used in our work is depicted in Figure 10, which is, except for some minor modifications, the network structure detailed in [91]. In the first layer of the network, we process separately the load information and, when available, the exogenous information such as temperature readings. Later, the results will be concatenated together and processed by a deep residual network with L layers. Each layer consists of a residual block with 1-D DCC, a rectified linear unit (ReLU) activation and finally dropout to prevent overfitting. The output layer consists of 1×1 convolution, which allows the network to output a one-dimensional vector $\mathbf{y} \in \mathrm{IR}^{n_T}$ having the same dimensionality as the input vector $\mathbf{x}$. To approach multi-step forecasting, we adopt a MIMO strategy.

6.3 | CNNs and TCNs' application for short-term load forecasting

In the short-term load forecasting relevant literature, CNNs have not been studied to a large extent. Indeed, until recently, these models were not considered for any time series-related problem. Still, several works tried to address the topic; in [15], a deep convolutional neural network model named DeepEnergy is presented. The proposed network is inspired by the first architectures used in ImageNet challenge (e.g. [81]), alternating convolutional and pooling layers, halving the width of the feature map after each step. According to the provided experimental results, DeepEnergy can precisely predict energy load in the next three days, outperforming five other machine learning algorithms including LSTM and FNN. In [16], a CNN is

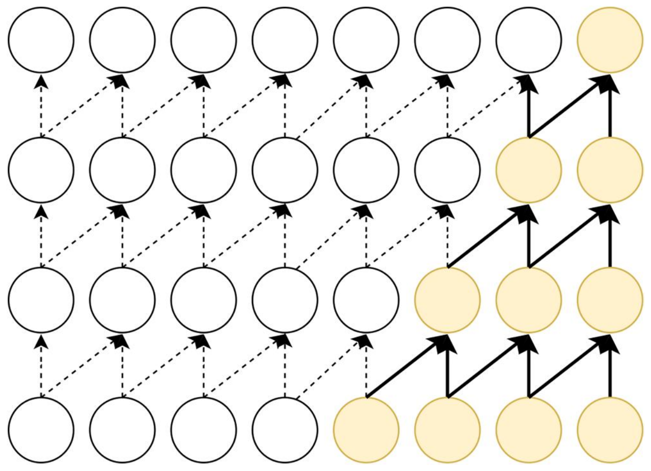

FIGURE 8 A three layers convolutional neural network with causal convolution (no dilation), the receptive field r is 4

compared to recurrent and feed-forward approaches, showing promising results on a benchmark dataset. In [25], a hybrid approach involving both convolutional and recurrent architectures is presented. The authors integrate different input sources and use convolutional layers to extract meaningful features from the historic load while the recurrent network's main task is to learn the system's dynamics. The model is evaluated on a large dataset containing hourly loads from a city in North China and is compared with a three-layer feed-forward neural network. A different hybrid approach is presented in [26]; the authors process the load information in parallel with a CNN and an LSTM. The features generated by the two networks are then used as an input for a final prediction network (fully connected) in charge of forecasting the day-ahead load. More recently, temporal convolutional networks have also started to be used and show very promising results indeed. In [39], a modified encoder–decoder architecture based on WaveNet is proposed. The authors performed careful testing of their model against the state-of-the-art power consumption data coming from the French grid. TCNs have been also used in probabilistic frameworks [40] to estimate probability densities in both parametric and non-parametric settings. The authors showed the superior performance of their method in both probabilistic forecasting and point estimates.

7 | PERFORMANCE ASSESSMENT

In this section, we perform evaluation and assessment of all the presented architectures. The testing is carried out by means of five use cases that are based on three different datasets. As said in the introduction, our goal is to directly compare and assess relevant deep learning models on standardised publicly available benchmarks so as to fully support experiment reproducibility; also, the designed code is free and has been made available to researchers and practitioners.[1]

We first introduce the performance metrics that we considered for both network optimisation and testing, then describe the datasets that have been used and finally we discuss the results.

7.1 | Performance metrics

The efficiency of the considered architectures has been measured and quantified using widely adopted error metrics. Specifically, we adopted the root mean squared error (RMSE) and the mean absolute error (MAE):

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{n_O}\sum_{t=0}^{n_O-1}\left(\hat{y}_i[t]-y_i[t]\right)^2}$$

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{n_O}\sum_{t=0}^{n_O-1}\left|\hat{y}_i[t]-y_i[t]\right|$$

where N is the number of input–output pairs provided to the model in the course of testing, $y_i[t]$ and $\hat{y}_i[t]$, respectively, are the real load values and the estimated load values at time t for sample i (i.e. the $i-th$ time window). $\langle\cdot\rangle$ is the mean operator, $\|\cdot\|_2$ is the Euclidean L2 norm, while $\|\cdot\|_1$ is the L1 norm. $\mathbf{y}\in \mathrm{IR}^{n_O}$ and $\hat{\mathbf{y}}\in \mathrm{IR}^{n_O}$ are the real load values and the estimated load values for one sample, respectively. Still, a more intuitive and indicative interpretation of the prediction efficiency of the estimators can be expressed by the normalised root mean squared error, which, different from the above two metrics, is independent from the scale of the data:

$$\mathrm{NRMSE}_{\%} = \frac{\mathrm{RMSE}}{y_{max}-y_{min}}\cdot 100$$

where y_{max} and y_{min} are the maximum and minimum value of the training dataset, respectively. In order to quantify the proportion of variance in the target that is explained by the forecasting methods, we also consider the R^2 index:

$$\mathrm{R}^2 = \frac{1}{N}\sum_{i=0}^{N-1}\left(1-\frac{\mathrm{RSS}}{\mathrm{TSS}}\right)$$

$$\mathrm{RSS} = \frac{1}{n_O}\sum_{t}^{n_O}\left(\hat{y}_i[t]-y_i[t]\right)^2 \qquad \mathrm{TSS} = \frac{1}{n_O}\sum_{t}^{n_O}\left(y_i[t]-\bar{y}_i\right)^2$$

where $\bar{y}_i = \frac{1}{n_O}\sum_{t}^{n_O} y_i[t]$

All considered models have been implemented in Keras 2.12 [94] with Tensorflow [95] as backend. The experiments are executed on a Linux cluster with an Intel(R) Xeon(R) Silver CPU and an Nvidia Titan XP.

[1] https://github.com/albertogaspar/dts

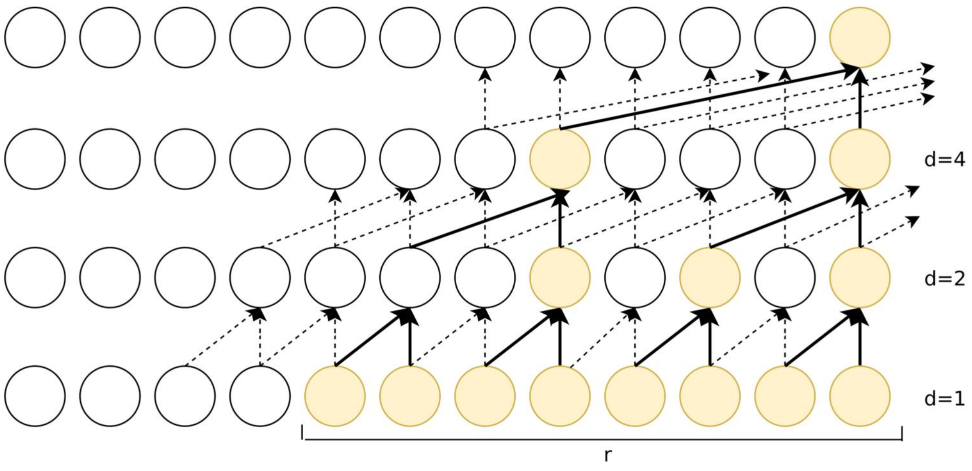

FIGURE 9 A three layers convolutional neural network with dilated causal convolutions. The dilation factor d grows on each layer by a factor of two and the kernel size k is 2, thus the output neuron is influenced by eight input neurons, that is,, the history size is 8

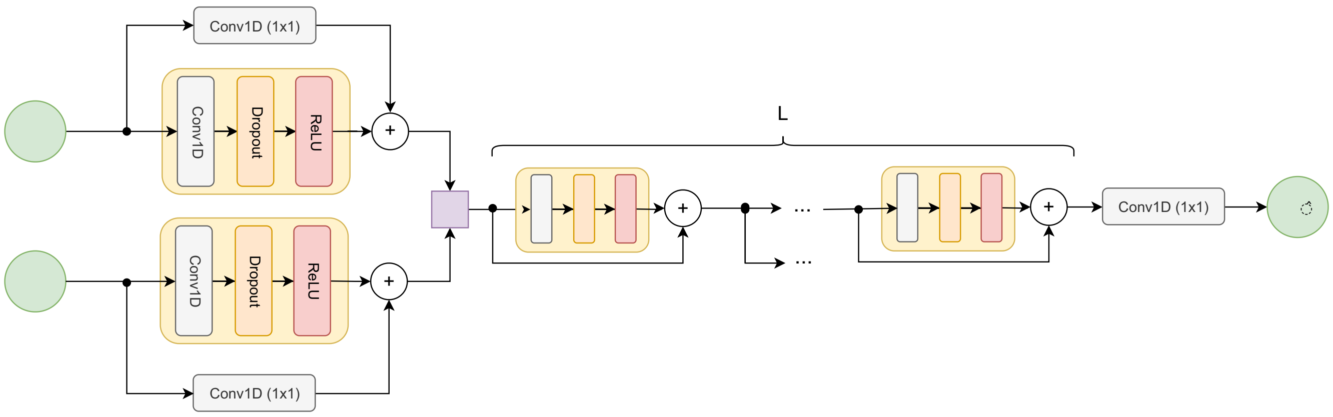

FIGURE 10 Temporal convolutional network Architecture. $\mathbf{x_t}$ is the vector of historical loads along with the exogenous features for the time window indexes from 0 to n_T, $\mathbf{z_t}$ is the vector of exogenous variables related to the last n_O indexes of the time window (when available), $\hat{\mathbf{y}}_t$ is the output vector. Residual Blocks are composed by a 1D Dilated Causal Convolution, a ReLU activation and Dropout. The square box represents a concatenation between (transformed) exogenous features and (transformed) historical loads

7.2 | Use case I, II and III (individual households)

In this scenario, we aim at testing and validating the usability of the considered models in the case of inputs with very challenging and noisy dynamics. The first use case considers the individual household electric power consumption dataset (IHEPC), which contains 2.07 M measurements of electric power consumption for a single house located in Sceaux (7 km of Paris, France). Measurements are collected every minute between December 2006 and November 2010 (47 months) [29]. In this study, we focus on predicting the ‘Global active power’ parameter. Nearly 1.25% of measurements are missing, still, all the available ones come with timestamps. We reconstruct the missing values using the mean power consumption for the corresponding time slot across the different years of measurements. In order to have a unified approach, we have decided to resample the dataset using a sampling rate of 15 min, which is a widely adopted standard in modern smart metre technologies. In Table 3, the sample size is outlined for each dataset.

In this use case, we performed the forecasting using only historical load values as relevant exogenous variables were not available. The right side of Figure 11 depicts the average weekly electric consumption. As expected, it can be observed that the highest consumption is registered in the morning and evening periods of the day when the occupancy of resident houses is high. Moreover, the average load profile over a week clearly shows that weekdays are similar while weekends present a different trend of consumption.

The figure shows that the data are characterised by high variance. The prediction task consists in forecasting the electric load for the next day, that is, 96 timesteps ahead.

In order to assess the performance of the architectures, we hold out a portion of the data that denotes our test set and comprises the last year of measurements. The remaining measurements are repeatedly divided in two sets, keeping aside a month of data every five months. This process allows us to build a training set and a validation set on which different hyperparameter configurations can be evaluated. Only the best performing configuration is later evaluated on the test set.

The second and third use cases consider the data coming from the smart metring electricity customer behaviour trials (CBTs), which took place during 2009 and 2010 with over 5000 Irish homes and businesses participating [96]. The data are collected and made available by the Commission for Energy Regulation (CER), which is the regulator for the electricity and natural gas sectors in Ireland. Since we are interested in evaluating the performance of our model on a single household, we selected 2 m (2113 and 4088) that, by showing a different consumption profile, become reference instances of the household class. In this way, reproducibility of results is granted; yet, some variability is considered. More household instances can be indeed considered but are outside the scope of this work. In this use case, as with the previous one, we performed the forecasting using only historical load values as no exogenous information is available. The forecasting horizon is still one day and the preprocessing method and the model selection criteria used are also the same. We report the average weekly electric consumption in Figure 12 for both considered metres. By looking at these images, we can immediately notice that both metres display high variance compared to use case I. In particular, metre 4088 presents a huge difference between its mean and median, which anticipates that prediction for this use case would be challenging.

7.3 | Use cases IV and V (aggregated load)

As the accurate consumption forecast of an area (e.g., supplied by a feeder, substation or even entire town) represents a very relevant problem in smart grids, in our assessment, we devote special attention to this scenario. We believe that the main contribution of our work to research and industry practice can actually be related to providing a clear insight into the usability of the proposed models to such issues. The other two use cases are based on the GEFCom2014 dataset [35], which was made available for an online forecasting competition that lasted between August 2015 and December 2015. The dataset contains 60.6k hourly measurements of (aggregated) electric power consumption collected by ISO New England between January 2005 and December 2011. Different from the dataset analysed before, temperature values are also available and are used by the different architectures to enhance their prediction performance. In particular, the input variables used for forecasting the subsequent n_O at timestep t include several previous load measurements, the temperature measurements for the previous timesteps registered by 25 different stations, hour, day, month and year of the measurements. We apply standard normalisation to load and temperature measurements while for other variables we simply apply one-hot encoding, that is, we code information in K-dimensional vector in which one of the elements equals 1, and all others equal 0 [97]. On the right side of Figure 11, we observe the average load and data dispersion on a weekly basis. Compared to IHEPC and CERs, the load profiles here look much more regular. This meets intuitive expectations as the load measurements in the previous datasets come from a single household; thus, the randomness introduced by the user behaviour has a more remarkable impact on the results. On the contrary, the load information in GEFCom2014 comes from the aggregation of the data provided by several different smart metres; clustered data exhibits a more stable and regular pattern. The main task of these use cases, as well the previous one, consists in forecasting the electric load for the next day, that is, 24 timesteps ahead. The hyperparameters' optimisation and the final score for the models follow the same guidelines provided for use cases I, II and III; the number of points for each subset is described in Table 3.

TABLE 3 Sample size of training, validation and test sets for each dataset

Dataset	Train	Test
IHEPC	103,301	35,040
CER – 2113	21,406	4320
CER – 4088	21,406	4320
GEFCom2014	44,640	8928

Abbreviation: CER, Commission for Energy Regulation.

7.4 | Results

The compared architectures are the ones presented in the previous sections with one exception. In fact, we have additionally considered a deeper variant of a feed-forward neural network with residual connections, which is named DFNN in this work. In accordance with [98], we have employed a 2-shortcut network, that is, the input undergoes two affine transformations each followed by a non-linearity before being summed to the original values. For regularisation purposes, we have included dropout and batch normalisation [99] in each residual block. We have additionally inserted this model in the results' comparison as it represents an evolution of standard feed-forward neural networks, which is expected to better handle highly complex time series data.

Moreover, to allow comparison between deep learning models and standard time series analysis technique, we additionally consider an ARIMA model in the experimental evaluation (AR, ARMA or ARMAX models are not worth considering here as none of the considered time series is stationary).

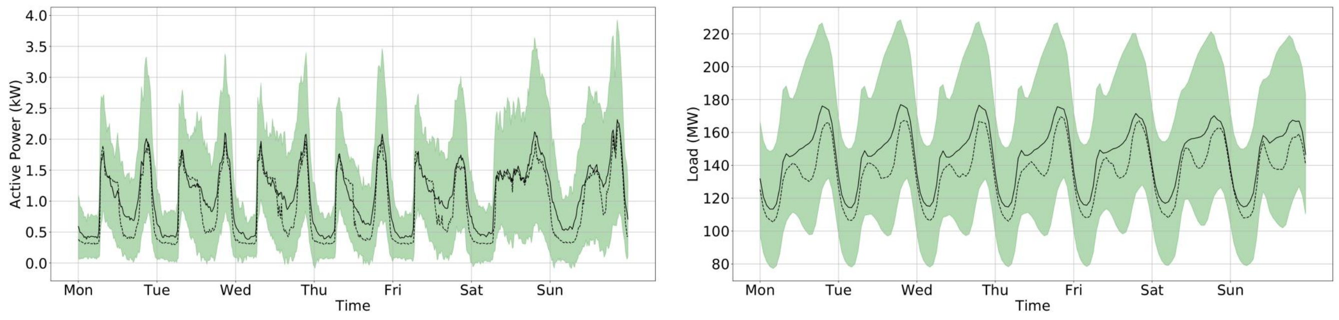

FIGURE 11 Weekly statistics for the electric load in the whole IHEPC (Left) and GEFCom2014 datasets (right). The bold line is the mean curve, the dotted line is the median and the green area covers one standard deviation from the mean

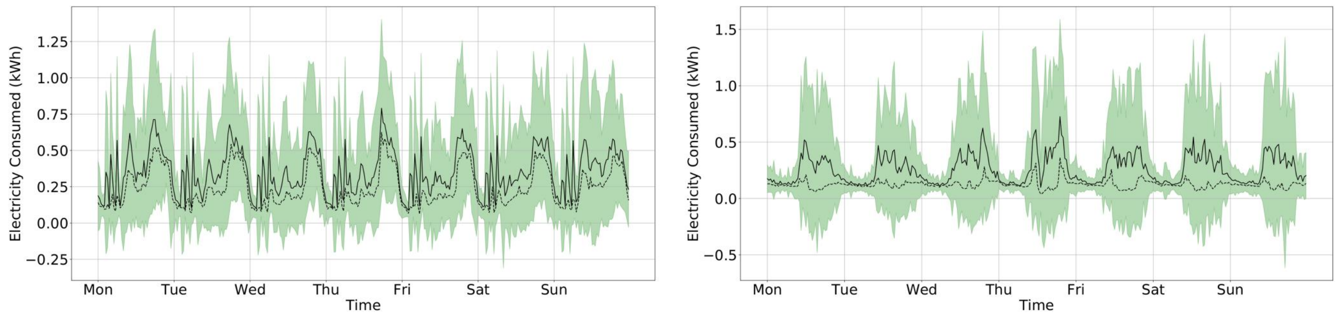

FIGURE 12 Weekly statistics for the electric load reported by smart metres 2113 (right) and 4088 (left) in the commission for energy regulation dataset. The bold line is the mean curve, the dotted line is the median and the green area covers one standard deviation from the mean

Table 4 summarises the best configurations found through grid search for each model and use case. For all datasets, we experimented different input sequences of length n_T. Finally, we used a window size of four days, which has been found to be the best trade-off between performance and memory requirements. The output sequence length n_O is fixed to one day. For each model, we identified the optimal number of stacked layers in the network L, the number of hidden units per layer n_H, the regularisation coefficient λ (L2 regularisation) and the dropout rate p_d. Moreover, for TCN, we additionally tuned the width k of the convolutional kernel and the number of filters applied at each layer M (i.e., the depth of each output volume after the convolution operation). The dilation factor is increased exponentially with the depth of the network, that is, $d = 2^{\ell}$ with ℓ being the $\ell - th$ layer of the network.

Tables 5 to 7 summarise the test scores of the presented architectures obtained for the IHEPC dataset, the CER dataset with metre id 2113 and the one with metre id 4088, respectively. Certain similarities among networks trained for different use cases can be spotted out already at this stage. In particular, we observe that all models exploit a small number of neurons. This is not usual in deep learning but—at least for recurrent architectures—is consistent with [18].

Among recurrent neural networks, we observe that, in general, the MIMO strategy outperforms the recursive one in this multi-step prediction task. This is reasonable in the individual household scenario. Indeed, the recursive strategy, different from the MIMO one, is highly sensitive to error accumulation, which, in a highly volatile time series as the ones addressed here, results in a very inaccurate forecast. Among the MIMO models, we observe that in some cases (see Table 5) gated networks perform slightly better than the simple Elmann network, but this is not a common pattern among the investigated datasets. Thus, for customer-level load forecasting, there is no sufficient evidence to claim the superiority of gated systems. In general, we notice that all models, except the RNNs trained with a recursive strategy, achieve comparable performance and none really stands out. It is interesting to comment that recurrent networks outperform sequence-to-sequence architectures, which are supposed to better model complex temporal dynamics like the one exhibited by the residential load curve. Nevertheless, by observing the performance of recurrent networks trained with the recursive strategy, this behaviour is less surprising. In fact, compared with the aggregated load profiles, the load curve belonging to a single smart metre is way more volatile and sensitive to customer behaviour. For this reason, leveraging geographical and socio-economic features that characterise the area where the user lives may allow deep networks to generate better predictions. The statement above is valid for the IHEPC dataset and CER – 2113, while for CER – 4088 the described behaviours are less evident. This comes from the last use case being dominated by noise, and as such, being hardly predictable. On the contrary, metre 2113 and IHEPC have more predictable patterns and indeed, by looking at the results tables

TABLE 4 Best configurations found via grid search for all dataset used

					ERNN		LSTM		GRU		seq2seq	
Hyperparameters	**Dataset**	**FNN**	**DFNN**	**TCN**	**Rec**	**MIMO**	**Rec**	**MIMO**	**Rec**	**MIMO**	**TF**	**SG**
L	IHPEC	3	6	8	3	1	2	1	2	1	1	1
	CER – 2113	3	3	6	2	2	3	1	3	3	1	1
	CER – 4088	2	2	6	1	4	1	1	1	2	1	1
	GEFCOM	1	6	6	4	4	4	2	4	1	2	1
	GEFCOM$_{exog}$	1	6	8	2	1	4	1	2	2	2	3
n_H	IHPEC	256 - 128$_{x2}$	50	-	10	30	20	20	10	50	30	50
	CER – 2113	50	50	-	10	10	10	10	10	10	10	10
	CER – 4088	20	10	-	15	10	15	10	10	15	10	10
	GEFCOM	60	30	-	20	50	15	20	30	20	10	50
	GEFCOM$_{exog}$	60	30	-	10	30	30	50	10	15	20	20-15-10
λ	IHPEC	0.001	0.0005	0.005	0.001	0.001	0.001	0.001	0.001	0.0005	0.01	0.01
	CER – 2113	0.001	0.001	0.01	0.005	0.001	0.005	0.001	0.005	0.001	0.01	0.01
	CER – 4088	0.001	0.005	0.01	0.01	0.001	0.001	0.001	0.001	0.001	0.01	0.01
	GEFCOM	0.01	0.0005	0.01	0.01	0.0005	0.001	0.001	0.01	0.0005	0.01	0.01
	GEFCOM$_{exog}$	0.005	0.0005	0.005	0.0005	0.001	0.0005	0.0005	0.001	0.01	0.001	0.01
p_{drop}	IHPEC	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2
	CER – 2113	0.1	0.2	0.1	0.2	0.2	0.1	0.1	0.2	0.0	0.2	0.1
	CER – 4088	0.2	0.2	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.1	0.1
	GEFCOM	0.1	0.1	0.1	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.1
	GEFCOM$_{exog}$	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0
k, M	IHPEC	-	-	2, 32	-	-	-	-	-	-	-	-
	CER – 2113	-	-	2, 8	-	-	-	-	-	-	-	-
	CER – 4088	-	-	2, 8	-	-	-	-	-	-	-	-
	GEFCOM	-	-	2, 16	-	-	-	-	-	-	-	-
	GEFCOM$_{exog}$	-	-	2, 64	-	-	-	-	-	-	-	-

Abbreviations: CER, Commission for Energy Regulation; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MIMO, Multiple input-Multiple output; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

one can immediately notice how models' performance are distributed on a much wider range. For use cases I and II a classical method like ARIMA is easily outperformed by all considered deep learning techniques. Once again, this is not true for use case III for reasons explained above. For visualisation purposes, we compare all the models' performance for IHEPC. On the left side of Figure 13, we outline a single-day prediction scenario while on the right side of Figure 13, we quantify the differences between the best predictor (the GRU-MIMO) and the actual measurements; the thinner the line, the closer the prediction to the true data. Furthermore, in this figure, we concatenate multiple day predictions to have a wider time span and evaluate the model predictive capabilities. We observe that the model is able to generate a prediction that correctly models the general trend of the load curve but fails to predict steep peaks. This might come from the design choice of using MSE as the optimisation metric, which could discourage deep models from predicting high peaks as large errors are hugely penalised, and therefore, predicting a lower and smoother function results in better performance according to this metric. Alternatively, some of the peaks may simply represent noise due to a particular user behaviour and are thus unpredictable by definition.

The load curve of the second dataset (GEFCom2014) results from the aggregation of several different load profiles producing a smoother load curve when compared with the individual load case. Hyperparameters' optimisation and the final score for the models can be found in Table 4.

Table 8 and Table 9 show the experimental results obtained by the models in two different scenarios. In the former case,

TABLE 5 Individual household electric power consumption dataset results

		RMSE	**MAE**	**NRMSE**	R^2
FNN		0.76 ± 0.01	0.53 ± 0.01	10.02 ± 0.17	0.250 ± 0.026
DFNN		0.75 ± 0.01	0.53 ± 0.01	9.90 ± 0.05	0.269 ± 0.007
TCN		0.76 ± 0.01	0.54 ± 0.00	10.07 ± 0.11	0.245 ± 0.017
ERNN	MIMO	0.79 ± 0.00	0.56 ± 0.00	10.33 ± 0.08	0.201 ± 0.012
	Rec	0.88 ± 0.02	0.69 ± 0.03	11.61 ± 0.29	0.001 ± 0.039
LSTM	MIMO	0.75 ± 0.00	0.53 ± 0.00	9.85 ± 0.04	0.276 ± 0.006
	Rec	0.84 ± 0.06	0.60 ± 0.07	11.06 ± 0.74	0.085 ± 0.125
GRU	MIMO	**0.75 ± 0.00**	**0.52 ± 0.00**	**9.83 ± 0.03**	**0.279 ± 0.004**
	Rec	0.89 ± 0.02	0.70 ± 0.02	11.64 ± 0.23	0.00 ± 0.04
seq2seq	TF	0.78 ± 0.01	0.57 ± 0.02	10.22 ± 0.17	0.221 ± 0.026
	SG	0.76 ± 0.01	0.53 ± 0.01	10.00 ± 0.14	0.253 ± 0.03
ARIMA		1.22 ± 0.00	1.49 ± 0.00	11.3 ± 0.00	0.057 ± 0.00

Note: Each model's mean score (± one standard deviation) comes from 10 repeated training processes.

Abbreviations: ARIMA, autoregressive-integrated moving average; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MAE, mean absolute error; MIMO, Multiple input-Multiple output; RMSE, root mean squared error; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

TABLE 6 CER results (metre 2113)

		RMSE	**MAE**	**NRMSE**	R^2
FNN		0.34 ± 0.00	0.22 ± 0.00	10.39 ± 0.00	0.137 ± 0.001
DFNN		0.34 ± 0.00	0.22 ± 0.00	10.41 ± 0.00	0.134 ± 0.002
TCN		**0.33 ± 0.00**	**0.21 ± 0.00**	**10.15 ± 0.02**	**0.175 ± 0.004**
ERNN	MIMO	0.35 ± 0.00	0.23 ± 0.00	10.47 ± 0.04	0.123 ± 0.007
	Rec	0.37 ± 0.00	0.27 ± 0.01	11.20 ± 0.05	−0.003 ± 0.009
LSTM	MIMO	0.35 ± 0.00	0.24 ± 0.00	10.60 ± 0.05	0.100 ± 0.009
	Rec	0.37 ± 0.00	0.27 ± 0.01	11.21 ± 0.07	−0.004 ± 0.011
GRU	MIMO	0.35 ± 0.00	0.24 ± 0.00	10.57 ± 0.10	0.105 ± 0.017
	Rec	0.37 ± 0.00	0.28 ± 0.01	11.28 ± 0.09	−0.017 ± 0.016
seq2seq	TF	0.38 ± 0.01	0.29 ± 0.01	11.43 ± 0.17	−0.046 ± 0.031
	SG	0.37 ± 0.00	0.26 ± 0.00	11.12 ± 0.00	0.011 ± 0.00
ARIMA		0.37 ± 0.00	0.26 ± 0.00	11.40 ± 0.00	−0.029 ± 0.00

Note: Each model's mean score (± one standard deviation) comes from 10 repeated training processes.

Abbreviations: ARIMA, autoregressive-integrated moving average; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MAE, mean absolute error; MIMO, Multiple input-Multiple output; RMSE, root mean squared error; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

only load values were provided to the models while in the latter scenario the input vector has been augmented with the exogenous features described before. Compared to the previous dataset, this time series exhibits a much more regular pattern; as such we expect the prediction task to be easier. Indeed, we can observe a major improvement in terms of performance across all the models.

As already noted in [22, 100], the prediction accuracy increases significantly when the forecasting task is carried out on a smooth load curve (resulting from the aggregation of many individual consumers).

We can observe that, in general, all models except ARIMA and plain FNNs benefit from the presence of exogenous variables. When exogenous variables are adopted, we notice a

TABLE 7 CER results (metre 4088)

		RMSE	MAE	NRMSE	R^2
FNN		**0.44 ± 0.00**	**0.23 ± 0.00**	**9.06 ± 0.00**	**0.079 ± 0.00**
DFNN		0.44 ± 0.00	0.23 ± 0.00	9.07 ± 0.00	0.077 ± 0.00
TCN		0.44 ± 0.00	0.23 ± 0.00	9.09 ± 0.02	0.074 ± 0.004
ERNN	MIMO	0.45 ± 0.01	0.24 ± 0.01	9.25 ± 0.14	0.040 ± 0.028
	Rec	0.47 ± 0.00	0.28 ± 0.02	8.95 ± 1.22	−0.008 ± 0.009
LSTM	MIMO	0.45 ± 0.00	0.23 ± 0.00	9.07 ± 0.01	0.078 ± 0.003
	Rec	0.46 ± 0.00	0.26 ± 0.02	9.43 ± 0.02	0.003 ± 0.0.004
GRU	MIMO	0.44 ± 0.00	0.23 ± 0.00	9.06 ± 0.01	0.078 ± 0.003
	Rec	0.47 ± 0.02	0.28 ± 0.03	9.66 ± 0.49	−0.050 ± 0.111
seq2seq	TF	0.46 ± 0.01	0.28 ± 0.02	9.37 ± 0.10	0.014 ± 0.022
	SG	0.45 ± 0.01	0.28 ± 0.01	9.17 ± 0.02	0.079 ± 0.00
ARIMA		0.46 ± 0.00	0.25 ± 0.00	9.4 ± 0.00	0.010 ± 0.00

Note: Each model's mean score (± one standard deviation) comes from 10 repeated training processes

Abbreviations: ARIMA, autoregressive-integrated moving average; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MAE, mean absolute error; MIMO, Multiple input-Multiple output; RMSE, root mean squared error; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

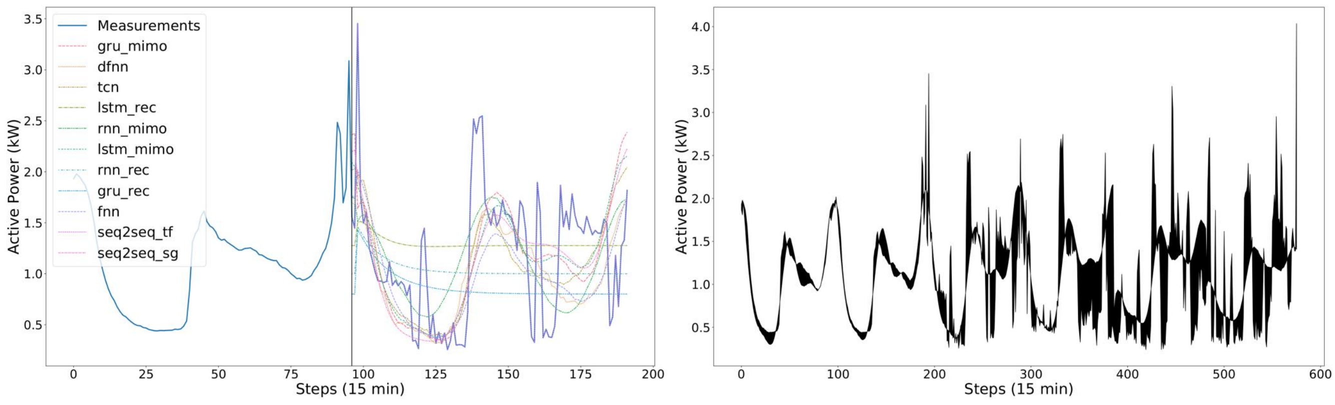

FIGURE 13 (Right) Predictive performance of all the models on a single day for IHEPC dataset. The left portion of the image shows (part of) the measurements used as input while the right side with multiple lines represents the different predictions. (Left) Difference between the best model's predictions (gated recurrent units-multiple input-multiple output) and the actual measurements. The thinner the line the closer the prediction is to the true data

major improvement by RNNs trained with the recursive strategy, which outperform MIMO ones. This increase in accuracy can be attributed to a better capacity of leveraging the exogenous time series of temperatures to yield a better load forecast. Moreover, RNNs with the MIMO strategy gain negligible improvement compared to their performance when no extra feature is provided. This kind of architectures use a feed-forward neural network to map their final hidden state to a sequence of n_O values, that is, the estimates. Exogenous variables are elaborated directly by this FNN, which, as observed above, have problems in handling both load data and extra information. Consequently, a better way of injecting exogenous variables in the MIMO recurrent network needs to be found in order to provide a boost in prediction performance comparable to the one achieved by employing the recursive strategy.

For reasons that are similar to those discussed above, sequence-to-sequence models trained via *teacher forcing* (seq2seq-TF) experienced improvement when exogenous features were used. Still, seq2seq models trained in the free-running mode (seq2seq-SG) proves to be a valid alternative to standard seq2seq-TF producing high quality predictions in all use cases. The absence of a discrepancy between training and inference in terms of data generating distribution shows to be an advantage as seq2seq-SG is less sensitive to noise and error propagation.

Finally, we notice that TCNs perform well in all the presented use cases. Considering their lower memory

TABLE 8 GEFCom2014 results without any exogenous variable. Each model's mean score (± one standard deviation) comes from 10 repeated training processes

		RMSE	**MAE**	**NRMSE**	R^2
FNN		21.1 ± 2.5	15.5 ± 2.1	7.01 ± 0.82	0.833 ± 0.041
DFNN		22.4 ± 6.2	17.1 ± 6.2	7.44 ± 2.01	0.801 ± 0.124
TCN		17.2 ± 0.1	11.5 ± 0.1	5.71 ± 0.14	0.891 ± 0.00
ERNN	MIMO	18.0 ± 0.3	11.9 ± 0.4	5.99 ± 0.11	0.879 ± 0.046
	Rec	27.0 ± 2.3	20.7 ± 2.5	8.95 ± 0.78	0.732 ± 0.046
LSTM	MIMO	19.5 ± 0.5	13.7 ± 0.6	6.47 ± 0.18	0.861 ± 0.007
	Rec	25.6 ± 2.2	18.4 ± 1.3	8.52 ± 0.72	0.757 ± 0.041
GRU	MIMO	19.0 ± 0.2	13.1 ± 0.3	6.29 ± 0.07	0.868 ± 0.003
	Rec	26.7 ± 3.3	19.8 ± 3.1	8.85 ± 1.09	0.737 ± 0.064
seq2seq	TF	21.5 ± 2.1	15.4 ± 1.9	7.13 ± 0.69	0.829 ± 0.034
	SG	**17.1 ± 0.2**	**11.3 ± 0.2**	**5.67 ± 0.06**	**0.893 ± 0.002**
ARIMA		30.1 ± 0.0	22.6 ± 0.0	10.00 ± 0.00	0.660 ± 0.00

Abbreviations: ARIMA, autoregressive-integrated moving average; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MAE, mean absolute error; MIMO, Multiple input-Multiple output; RMSE, root mean squared error; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

TABLE 9 GEFCom2014 results with exogenous variables

		RMSE	**MAE**	**NRMSE**	R^2
FNN		27.9 ± 2.8	20.8 ± 2.4	9.28 ± 0.93	0.709 ± 0.062
DFNN		23.0 ± 1.2	15.6 ± 0.7	7.62 ± 0.41	0.805 ± 0.021
TCN		15.4 ± 1.5	8.6 ± 1.7	5.00 ± 0.22	0.917 ± 0.007
ERNN	MIMO	17.9 ± 0.3	11.7 ± 0.3	5.94 ± 0.01	0.883 ± 0.004
	Rec	14.7 ± 1.0	8.6 ± 1.0	4.88 ± 0.19	0.925 ± 0.005
LSTM	MIMO	18.1 ± 1.3	12.1 ± 1.3	6.01 ± 0.42	0.877 ± 0.018
	Rec	**13.8 ± 0.6**	**7.5 ± 0.3**	**4.59 ± 0.18**	**0.930 ± 0.006**
GRU	MIMO	17.8 ± 0.2	11.7 ± 0.2	5.93 ± 0.07	0.882 ± 0.002
	Rec	16.7 ± 0.5	10.0 ± 0.6	5.54 ± 0.15	0.898 ± 0.006
seq2seq	TF	14.3 ± 1.0	8.5 ± 0.9	4.74 ± 0.32	0.924 ± 0.014
	SG	15.9 ± 1.8	9.8 ± 1.8	5.28 ± 0.60	0.907 ± 0.021
ARIMA		36.5 ± 0.0	26.7 ± 0.0	12.10 ± 0.00	0.503 ± 0.00

Note: Each model's mean score (± one standard deviation) comes from 10 repeated training processes

Abbreviations: ARIMA, autoregressive-integrated moving average; DFNN, deeper variant of a feed-forward neural network; ERNN, Elmann recurrent neural networks; FNN, feed-forward neural network; GRU, Gated recurrent units; LSTM, long short-term memory; MAE, mean absolute error; MIMO, Multiple input-Multiple output; RMSE, root mean squared error; SG, self-generated; TCN, temporal convolutional network; TF, teacher forcing.

requirements in the training process along with their inherent parallelism, this type of networks represents a promising alternative to recurrent neural networks for short-term load forecasting.

As a final note, we can observe that the ARIMA model is outperformed by all other approaches. This confirms that, also for aggregated load consumption, deep learning approaches are much more effective than classical techniques due to the presence of non-linearities.

The prediction results are presented in the same fashion as the previous use case in Figure 14. Observe that, in general, all considered models are able to produce reasonable estimates as sudden picks in consumption are smoothed. Therefore, predictors greatly improve their

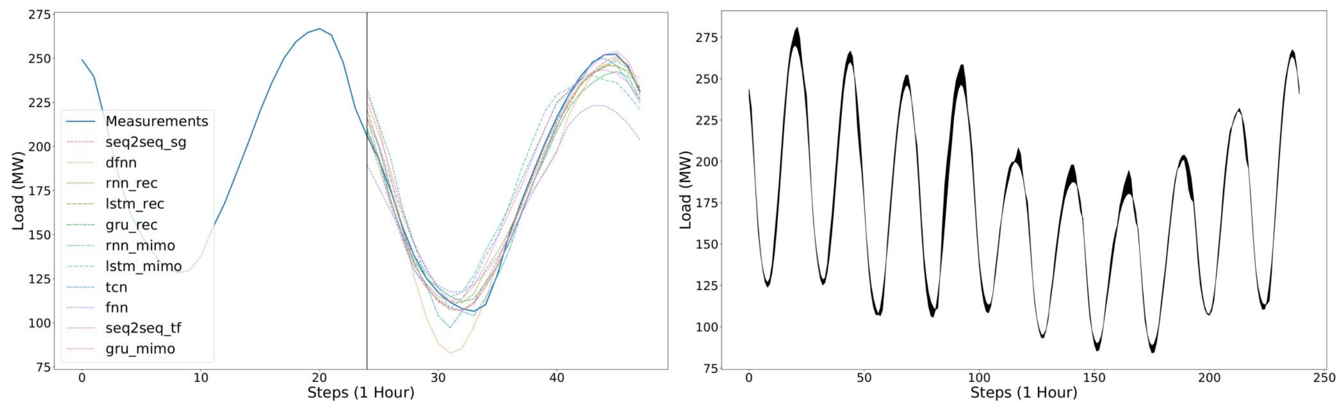

FIGURE 14 (Right) Predictive performance of all the models on a single day for GEFCom2014 dataset. The left portion of the image shows (part of) the measurements used as input while the right side with multiple lines represents the different predictions. (Left) Difference between the best model's predictions (long short-term memory-Rec) and the actual measurements. The thinner the line the closer the prediction to the true data

accuracy when predicting day-ahead values for the aggregated load curves with respect to an individual household scenario.

8 | CONCLUSIONS

In this work, we have surveyed and experimentally evaluated the most relevant deep learning models applied to the short-term load forecasting problem, paving the way for standardised assessment and identification of the most optimal solutions in this field. The focus has been given to the three main families of models, namely, recurrent neural networks, sequence-to-sequence architectures and recently developed temporal convolutional neural networks. An architectural description along with a technical discussion on how multi-step ahead forecasting is achieved, has been provided for each considered model. Moreover, different forecasting strategies are discussed and evaluated, identifying advantages and drawbacks for each of them. The evaluation has been carried out on five real-world use cases that refer to two distinct scenarios for load forecasting. Indeed, three use cases deal with datasets coming from a single household characterised by noisy dynamics while the other two tackle the prediction of a load curve that represents aggregated metres, dispersed over a wide area of the grid. Our findings concerning the application of recurrent neural networks to short-term load forecasting, show that the simple ERNN performs comparably to gated networks such as GRU and LSTM when adopted in aggregated load forecasting. Thus, the less costly alternative provided by ERNN may represent the most effective solution in this scenario as it allows to reduce the training time without remarkable impact on prediction accuracy. A similar conclusion may be drawn for a single-household electric load forecasting, where only in one use case gated networks prove to be superior to Elmann ones. Sequence-to-sequence models have demonstrated to be quite efficient in load forecasting tasks even though they seem to fail in outperforming RNNs. In general, we can claim that seq2-seq architectures do not represent a golden standard in load forecasting as they do in other domains such as natural language processing. In addition, regarding this family of architectures, we have observed that teacher forcing may not represent the best solution for training seq2seq models on short-term load forecasting tasks. Despite being harder in terms of convergence, free-running models learn to handle their own errors, avoiding the discrepancy between training and testing that is a well-known issue for teacher forcing. It turns out to be worth the effort to further investigate the capabilities of seq2seq models trained with intermediate solutions such as *professor forcing*. Finally, we evaluated the recently developed temporal convolutional neural networks, which demonstrated convincing performance when applied to load forecasting tasks. Therefore, we strongly believe that the adoption of these networks for sequence modelling in the considered field is very promising (especially for aggregated loads) and might even introduce a significant advance in this area that is emerging as important for future smart grid development. We also comment that short-term load forecasting at the customer level has proved to be an extremely challenging task for deep learning models as well. As a future direction in this specific topic, we are interested in exploiting time series correlation to learn a single model for multiple metres that exhibits similar behaviour. Our preliminary results are promising [101] and we believe that such an approach may provide considerable improvement in the field compared to the approach presented here. We hope that the presented work as well as the open sourcing of a library for short-term load forecasting may stimulate other authors to contribute to the research in the field.

ACKNOWLEDGEMENTS

This project is carried out within the frame of the Swiss Centre for Competence in Energy Research on the Future Swiss Electrical Infrastructure (SCCER-FURIES) with the financial

support of the Swiss Innovation Agency (Innosuisse – SCCER program).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are derived from the following resources available in the public domain: https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption, http://blog.drhongtao.com/2017/03/gefcom2014-load-forecasting-data.html, https://www.ucd.ie/issda/data/commissionforenergyregulationcer/. Restrictions apply to the availability of these data, which were used under licence for this study.

ORCID

Alberto Gasparin https://orcid.org/0000-0003-3350-3168

REFERENCES

1. Fang, X., et al.: Smart grid – the new and improved power grid: a survey. IEEE Commun. Surv. Tutorials. 14(4), 944–980 (2012)
2. Almeshaiei, E., Soltan, H.: A methodology for electric power load forecasting. Alexandria Eng. J. 50(2), 137–144. (2011). http://www.sciencedirect.com/science/article/pii/S1110016811000330
3. Hippert, H.S., Pedreira, C.E., Souza, R.C.: Neural networks for short-term load forecasting: a review and evaluation. IEEE Trans Power Syst. 16(1), 44–55 (2001)
4. Chen, J.-F., Wang, W.-M., Huang, C.-M.: Analysis of an adaptive time-series autoregressive moving-average (ARMA) model for short-term load forecasting. Elec. Power Syst. Res. 34(3), 187–196 (1995)
5. Huang, S.-J., Shih, K.-R.: Short-term load forecasting via arma model identification including non-Gaussian process considerations. IEEE Trans. Power Syst. 18(2), 673–679 (2003)
6. Hagan, M.T., Behr, S.M.: The time series approach to short term load forecasting. IEEE Trans. Power Syst. 2(3), 785–791 (1987)
7. Huang, C.-M., Huang, C.-J., Wang, M.-L.: A particle swarm optimization to identifying the ARMAX model for short-term load forecasting. IEEE Trans. Power Syst. 20(2), 1126–1133 (2005)
8. Yang, H.-T., Huang, C.-M., Huang, C.-L.: Identification of ARMAX model for short term load forecasting: an evolutionary programming approach. In: Power Industry Computer Application Conference, 1995, pp. 325–330. IEEE (1995)
9. Newsham, G.R., Birt, B.J.: Building-level occupancy data to improve arima-based electricity use forecasts. In: Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-efficiency in Building, pp. 13–18. Zurich (2010)
10. Lee, K.Y., Cha, Y.T., Park, J.H.: Short-term load forecasting using an artificial neural network. IEEE Trans. Power Syst. 7(1), 124–132 (1992)
11. Park, D.C., et al.: Electric load forecasting using an artificial neural network. IEEE Trans. Power Syst. 6(2), 442–449 (1991)
12. Srinivasan, D., Liew, A., Chang, C.: A neural network short-term load forecaster. Elec. Power Syst. Res. 28(3), 227–234. (1994). http://www.sciencedirect.com/science/article/pii/037877969490037X
13. Drezga, I., Rahman, S.: Short-term load forecasting with local ANN predictors. IEEE Trans. Power Syst. 14(3), 844–850 (1999)
14. Chen, K., et al.: Short-term load forecasting with deep residual networks. IEEE Trans. Smart Grid. 10(4), 3943–3952 (2018)
15. Kuo, P.-H., Huang, C.-J.: A high precision artificial neural networks model for short-term energy load forecasting. Energies. 11(1), 213. (2018). http://www.mdpi.com/1996-1073/11/1/213
16. Amarasinghe, K., Marino, D.L., Manic, M.: Deep neural networks for energy load forecasting. In: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), pp. 1483–1488. June Edinburgh (2017)
17. Nayaka, S., Yelamali, A., Byahatti, K.: Electricity short term load forecasting using Elman recurrent neural network. In: 2010 International Conference on Advances in Recent Technologies in Communication and Computing, vol. 11, pp. 351–354. Kottayam (2010)
18. Bianchi, F.M., et al.: Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis SpringerBriefs in Computer Science, 1 st edn. Spirnger International Publishing (2017). https://www.springer.com/gp/book/9783319703374
19. Bianchi, F.M., et al.: Short-term electric load forecasting using echo state networks and PCA decomposition. IEEE Access. 3, 1931–1943 (2015)
20. Mocanu, E., et al.: Deep learning for estimating building energy consumption. Sustain. Energy Grids Networks. 6, 91–99 (2016)
21. Zheng, J., et al.: Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network. In: 2017 51st Annual Conference on Information Sciences and Systems (CISS), pp. 1–6. IEEE Baltimore (2017)
22. Kong, W., et al.: Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Trans. Smart Grid. 10, 841–851 (2017)
23. Bouktif, S., et al.: Optimal deep learning LSTM model for electric load forecasting using feature selection and genetic algorithm: comparison with machine learning approaches. Energies. 11(7), 1636 (2018)
24. Wang, Y., et al.: Short-term load forecasting with multi-source data using gated recurrent unit neural networks. Energies. 11, 1138 (2018)
25. He, W.: Load forecasting via deep neural networks. Procedia Comput. Sci. 122, pp. 308–314. (2017). http://www.sciencedirect.com/science/article/pii/S1877050917326170
26. Tian, C., et al.: A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network. Energies. 11, 3493 (2018)
27. Hong, T., Pinson, P., Fan, S.: Global energy forecasting competition 2012. Int. J. Forecast. 30(2), 357–363. (2014). http://www.sciencedirect.com/science/article/pii/S0169207013000745
28. Marvuglia, A., Messineo, A.: Using recurrent artificial neural networks to forecast household electricity consumption. Energy Procedia. 14, pp. 45–55. (2012). http://www.sciencedirect.com/science/article/pii/S1876610211043116
29. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository. http://archive.ics.uci.edu/ml (2017). Accessed 15 Sept 2021
30. Smart grid, smart city, Australian govern, Australia, Canberray. http://www.industry.gov.au/ENERGY/PROGRAMMES/SMARTGRIDSMARTCITY/Pages/default.aspx. Accessed 15 Sept 2021
31. Cheng, Y., et al.: PowerLSTM: power demand forecasting using long short-term memory neural network. In: Cong, G., et al. (eds.) Advanced Data Mining and Applications, pp. 727–740. Springer International Publishing, Cham (2017)
32. Umass smart dataset. (2017). http://traces.cs.umass.edu/index.php/Smart/Smart. Accessed 15 Sept 2021
33. Marino, D.L., Amarasinghe, K., Manic, M.: Building energy load forecasting using deep neural networks. In: Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE, pp. 7046–7051. IEEE Florence (2016)
34. Wilms, H., Cupelli, M., Monti, A.: Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting. In: 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), pp. 673–679. IEEE Porto (2018)
35. Hong, T., et al.: Probabilistic energy forecasting: global energy forecasting competition 2014 and beyond. Int. J. Forecast. 32(3), 896–913. (2016). http://www.sciencedirect.com/science/article/pii/S0169207016000133
36. Jin, X.-B., et al.: Deep-learning forecasting method for electric power load via attention-based encoder-decoder with Bayesian optimization. Energies. 14(6), 1596 (2021)
37. Sehovac, L., Nesen, C., Grolinger, K.: Forecasting building energy consumption with deep learning: a sequence to sequence approach. In: 2019 IEEE International Congress on Internet of Things (ICIOT), pp. 108–116. Milan (2019)

38. Sehovac, L., Grolinger, K.: Deep learning for load forecasting: sequence to sequence recurrent neural networks with attention. IEEE Access. 8, 36411–36426 (2020)
39. Dorado Rueda, F., Durán Suárez, J., del Real Torres, A.: Short-term load forecasting using encoder-decoder wavenet: application to the French grid. Energies. 14(9), 2524 (2021)
40. Chen, Y., et al.: Probabilistic forecasting with temporal convolutional neural network. Neurocomputing. 399, 491–501 (2020)
41. Trindade, A.: ElectricityLoadDiagrams20112014 data set. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014# (2015). Accessed 15 Sept 2021
42. Almalaq, A., Edwards, G.: A review of deep learning methods applied on load forecasting. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 511–516. Cancun Dec 2017
43. Csáji, B.C.: Approximation with artificial neural networks. Faculty of Sciences, vol. 24, p. 7. Etvs Lornd University, Hungary (2001)
44. Hinton, G., Srivastava, N., Swersky, K.: Neural networks for machine learning. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed 15 Sept 2021
45. Zeiler, M.D.: Adadelta: an adaptive learning rate method. vol. 1212, pp. 12. (2012)
46. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego. 7–9 May 2015. http://arxiv.org/abs/1412.6980
47. Chen, S., Yu, D., Moghaddamjo, A.: Weather sensitive short-term load forecasting using nonfully connected artificial neural network. IEEE Trans. Power Syst. 3, 8 (1992)
48. He, K., et al.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, pp. 770–778. Las Vegas. 27–30 June 2016. https://doi.org/10.1109/CVPR.2016.90
49. Elman, J.L.: Finding structure in time. Cognit. Sci. 14(2), 179–211 (1990)
50. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. 9(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
51. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 1724–1734. Doha. 25–29 Oct 2014. http://aclweb.org/anthology/D/D14/D14-1179.pdf
52. Werbos, P.J.: Backpropagation through time: what it does and how to do it. In: Proceedings of the IEEE. vol. 78, pp. 1550 –1560 (1990). http://doi.org/10.1109/5.58337
53. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Parallel distributed processing: Explorations in the microstructure of cognition, In: Rumelhart D.E., McClelland J.L., C. PDP Research Group (eds.) ch. Learning Internal Representations by Error Propagation, vol. 1, pp. 318–362. MIT Press, Cambridge. (1986). http://dl.acm.org/citation.cfm?id=104279.104293
54. Williams, R.J., Peng, J.: An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Comput. 2, 490–501 (1998)
55. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. Trans. Neur. Netw. 5(2), 157–166 (1994). https://doi.org/10.1109/72.279181
56. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on International Conference on Machine Learning – vol. 28, pp. III–1310–III–1318. JMLR.org. Atlanta (2013). http://dl.acm.org/citation.cfm?id=3042817.3043083
57. Greff, K., et al.: LSTM: a search space odyssey. IEEE Trans. Neural Network. Learn. Syst. 28(10), 2222–2232 (2017)
58. Chung, J., et al.: Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, vol. abs/1412.3555 (2014)
59. Yin, W., et al.: Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923* (2017)
60. Pascanu, R., et al.: How to construct deep recurrent neural networks. In: Proceedings of the Second International Conference on Learning Representations (ICLR 2014) Banff (2014)
61. Schmidhuber, J.: Learning complex, extended sequences using the principle of history compression. Neural Comput. 4(2), 234–242 (1992). https://doi.org/10.1162/neco.1992.4.2.234
62. Graves, A., Mohamed, A.-R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. Vancouver (2013)
63. Hermans, M., Schrauwen, B.: Training and analysing deep recurrent neural networks. In: Burges, C.J.C., et al. (eds.) Advances in Neural Information Processing Systems 26, pp. 190–198. Curran Associates, Inc. Lake Tahoe (2013). http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf
64. Taieb, S.B., et al.: A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. Expert Syst. Appl. 39(8), 7067–7083. (2012). http://www.sciencedirect.com/science/article/pii/S0957417412000528
65. Sorjamaa, A., Lendasse, A.: Time series prediction using dirrec strategy. In: European Symposium on Artificial Neural Networks, vol. 6(01), pp. 143–148. Bruges (2006)
66. Bontempi, G.: Long term time series prediction with multi-input multi-output local learning. In: Proceedings of the 2nd European Symposium on Time Series Prediction (TSP) ESTSP08 (2008)
67. Taieb, S.B., et al.: Long-term prediction of time series by combining direct and mimo strategies. In: 2009 International Joint Conference on Neural Networks, pp. 3054–3061. (2009)
68. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, pp. 3104–3112. Montreal. 8–13 Dec 2014. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks
69. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. *CoRR*, vol. abs/1409.0473 (2014)
70. Wu, Y., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016)
71. Chorowski, J.K., et al.: Attention-based models for speech recognition. In: Cortes, C., et al. (eds.) Advances in Neural Information Processing Systems 28, pp. 577–585. Curran Associates, Inc. Montreal (2015). http://papers.nips.cc/paper/5847-attention-based-models-for-speech-recognition.pdf
72. Bahdanau, D., et al.: End-to-end attention-based large vocabulary speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4945–4949. Lujiazui, Mar (2016)
73. Xu, K., et al.: Show, attend and tell: neural image caption generation with visual attention. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 2048–2057. PMLR, Lille. (2015). http://proceedings.mlr.press/v37/xuc15.html
74. Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Comput. 1, 270–280 (1989)
75. Ranzato, M., et al.: Sequence level training with recurrent neural networks. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan. 2–4 May 2016. http://arxiv.org/abs/1511.06732
76. Bengio, S., et al.: Scheduled sampling for sequence prediction with recurrent neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems – vol. 1, pp. 1171–1179. MIT Press, Cambridge. (2015). http://dl.acm.org/citation.cfm?id=2969239.2969370
77. Lamb, A., et al.: Professor forcing: a new algorithm for training recurrent networks. In: NIPS Barcelona (2016)

78. Keneshloo, Y., et al.: Deep reinforcement learning for sequence-to-sequence models. IEEE Trans. Neural Network. Learn. Syst. 31(7), 2469–2489 (2020)
79. Wilms, H., Cupelli, M., Monti, A.: Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting. In: 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), pp. 673-679. https://doi.org/10.1109/INDIN.2018.8471953
80. Lecun, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE. 86, 2278–2324 (1998)
81. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., et al. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. Lake Tahoe (2012). http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
82. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego. 7–9 May 2015. http://arxiv.org/abs/1409.1556
83. Szegedy, C., et al.: Going deeper with convolutions. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June Boston (2015)
84. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448. Santiago (2015)
85. Ren, S., et al.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99. Montreal (2015)
86. Jaderberg, M., et al.: Spatial transformer networks. In: Advances in Neural Information Processing Systems, pp. 2017–2025. Montreal (2015)
87. Dahl, R., Norouzi, M., Shlens, J.: Pixel recursive super resolution. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 5449–5458. Venezia (2017)
88. Ledig, C., et al.: Photo-realistic single image super-resolution using a generative adversarial network. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 105–114. IEEE Honolulu (2017)
89. van den Oord, A., et al.: A generative model for raw audio. Arxiv. (2016). https://arxiv.org/abs/1609.03499
90. Gehring, J., et al.: Convolutional sequence to sequence learning. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 1243–1252. Sydney, 06–11 Aug 2017. http://proceedings.mlr.press/v70/gehring17a.html
91. Borovykh, A., Bohte, S., Oosterlee, K.: Conditional time series forecasting with convolutional neural networks. In: Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence, pp. 729–730. Sept (2017)
92. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. *CoRR*, vol. abs/160307285. (2016). http://arxiv.org/abs/1603.07285
93. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
94. Chollet, F., et al.: Keras. (2015). https://github.com/fchollet/keras
95. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. (2015). https://www.tensorflow.org/
96. Commission for Energy Regulation (CER). Cer smart metering project - electricity customer behaviour trial, 2009-2010. 1st edn. Irish Social Science Data Archive. SN: 0012-00. (2012). https://www.ucd.ie/issda/data/commissionforenergyregulationcer/
97. Bishop, C.M.: Pattern recognition and machine learning (information Science and Statistics). Springer-Verlag, Berlin (2006)
98. Li, S., et al.: Demystifying Resnet. *arXiv preprint arXiv:1611.01186* (2016)
99. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift, pp. 448–456. (2015). http://jmlr.org/proceedings/papers/v37/ioffe15.pdf
100. Marinescu, A., et al.: Residential electrical demand forecasting in very small scale: an evaluation of forecasting methods. In: 2013 2nd International Workshop on Software Engineering Challenges for the Smart Grid (SE4SG), pp. 25–32. San Francisco May (2013)
101. Cini, A., Lukovic, S., Alippi, C.: Cluster-based aggregate load forecasting with deep neural networks. In: 2020 International Joint Conference on Neural Networks (IJCNN) (to be published). IEEE Glasgow (2020)

How to cite this article: Gasparin, A., Lukovic, S., Alippi, C.: Deep learning for time series forecasting: the electric load case. CAAI Trans. Intell. Technol. 7(1), 1–25 (2022). https://doi.org/10.1049/cit2.12060