Neural Computing and Applications (2022) 34:16959–16971
https://doi.org/10.1007/s00521-022-07348-5

ORIGINAL ARTICLE

Long short-term cognitive networks

Gonzalo Nápoles[1] · Isel Grau[2] · Agnieszka Jastrzębska[3] · Yamisleydi Salgueiro[4]

Received: 4 October 2021 / Accepted: 26 April 2022 / Published online: 24 May 2022

Abstract
In this paper, we present a recurrent neural system named *long short-term cognitive networks* (LSTCNs) as a generalization of the short-term cognitive network (STCN) model. Such a generalization is motivated by the difficulty of forecasting very long time series efficiently. The LSTCN model can be defined as a collection of STCN blocks, each processing a specific time patch of the (multivariate) time series being modeled. In this neural ensemble, each block passes information to the subsequent one in the form of weight matrices representing the prior knowledge. As a second contribution, we propose a deterministic learning algorithm to compute the learnable weights while preserving the prior knowledge resulting from previous learning processes. As a third contribution, we introduce a feature influence score as a proxy to explain the forecasting process in multivariate time series. The simulations using three case studies show that our neural system reports small forecasting errors while being significantly faster than state-of-the-art recurrent models.

Keywords Short-term cognitive networks · Recurrent neural networks · Multivariate time series · Interpretability

1 Introduction

Time series analysis and forecasting techniques process data points that are ordered in a discrete-time sequence. While time series analysis focuses on extracting meaningful descriptive statistics of the data, time series forecasting uses a model for predicting the next value(s) of the series based on the previous ones. Traditionally, time series forecasting has been tackled with statistical techniques based on auto-regression or the moving average, such as exponential smoothing (ETS) Hyndman et al. [28] and the auto-regressive integrated moving average (ARIMA) Box et al. [8]. These methods are relatively simple and perform well in univariate scenarios and with relatively small data. However, they are more limited in predicting a long time horizon or dealing with multivariate scenarios.

The ubiquitousness of data generation in today's society brings the opportunity to exploit recurrent neural network (RNN) architectures for time series forecasting. RNN-based models have reported promising results in multivariate forecasting of long series Hewamalage et al. [25]. In contrast to feed-forward neural networks, RNN-based models capture long-term dependencies in the time sequence through their feedback loops. The majority of works published in this field are based on vanilla RNNs, Long-short Term Memory (LSTM) Hochreiter and Schmidhuber [26] or Gated Recurrent Unit (GRU) Cho et al. [13] architectures. In the last M4 forecasting competition Makridakis et al. [35], the winners were models combining RNNs with traditional forecasting techniques, such as exponential smoothing Smyl [50]. However, the use of RNN architectures is not entirely embraced by the forecasting community due to their lack of transparency,

✉ Gonzalo Nápoles
g.r.napoles@uvt.nl

Isel Grau
i.d.c.grau.garcia@tue.nl

Agnieszka Jastrzębska
A.Jastrzebska@mini.pw.edu.pl

Yamisleydi Salgueiro
yamisalgueiro@gmail.com

1 Department of Cognitive Science and Artificial Intelligence, Tilburg University, Tilburg, The Netherlands

2 Information Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands

3 Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland

4 Department of Computer Science, Universidad de Talca, Campus Curico, Talca, Chile

need for very specific configurations, and high computational cost [25, 35, 36].

In this regard, the development of RNN architectures for time series forecasting can bring serious financial and environmental costs. As anecdotal evidence, one of the participants in the last M4 forecasting competition reported getting a huge electricity bill from 5 computers running for 4.5 months Makridakis et al. [35]. More formally, the authors in Strubell et al [51] presented an eye-opening study characterizing the energy required to train recent deep learning models, including their estimated carbon footprint. An example of a training-intensive task is the tuning of the BERT model Devlin et al. [16] for natural language processing tasks, which compares to the CO_2 emissions of a trans-American flight. One of the conclusions of the study in Strubell et al. [51] is that researchers should focus on developing more efficient techniques and report measures (such as the training time) next to the model's accuracy.

A second concern related to the use of deep machine learning models is their lack of interpretability. For most high-stakes decision problems having an accurate model is insufficient; some degree of interpretability is also needed. There exist several model-agnostic post-hoc methods for computing explanations based on the predictions of a black-box model. For example, feature attribution methods such as SHAP Lundberg and Lee [34] approximate the Shapley values that explain the role of the features in the prediction of a particular instance. Other techniques such as LIME Ribeiro et al. [47] leverage the intrinsic transparency of other machine learning models (e.g., linear regression) to approximate the decisions locally. In contrast, intrinsically interpretable methods provide explanations from their structure and can be mappable to the domain Grau et al. [21]. In Rudin [48], the author argues that these explanations are more reliable and faithful to what the model computes. However, developing environmental-friendly RNN-based forecasting models able to provide a certain degree of transparency is a significant challenge.

In this paper, we propose the long short-term cognitive networks (LSTCNs) to cope with the efficient and transparent forecasting of long univariate and multivariate time series. LSTCNs involve a sequential collection of short-term cognitive network (STCN) blocks Nápoles et al. [39], each processing a specific time patch in the sequence. The STCN model allows for transparent reasoning since both weights and neurons map to specific features in the problem domain. Besides, STCNs allow for hybrid reasoning since the experts can inject knowledge into the network using prior knowledge matrices. As a second contribution, we propose a deterministic learning algorithm to compute the tunable parameters of each STCN block in a deterministic fashion. The highly efficient algorithm replaces the non-synaptic learning method presented in Nápoles et al. [39]. As a final contribution, we present a feature influence score as a proxy to explain the reasoning process of our neural system. The numerical simulations using three case studies show that our model produces high-quality predictions with little computational effort. In short, we have found that our model can be remarkably faster than state-of-the-art recurrent neural networks.

The rest of this paper is organized as follows. Section 2 revises the literature on time series forecasting with recurrent neural networks, while Sect. 3 presents the theory behind the STCN block. Section 4 is devoted to LSTCN's architecture, learning, and interpretability. Section 5 evaluates the performance of our model using three case studies involving long univariate and multivariate time series. Section 6 concludes the paper and provides future research directions.

2 Related work on time series forecasting

In the last decade, we observed a constantly growing share of artificial neural network-based approaches for time series forecasting. Prominent studies, including Bhaskar and Singh [7] and Ticknor [53], use traditional feed-forward neural architectures trained with the backpropagation algorithm for time series prediction. However, in more recent papers, we see a shift toward other neural models. In particular, RNNs have gained momentum Kong et al. [29].

Feed-forward neural networks consist of layers of neurons that are one-way connected, from the input to the output layer, without cycles. In contrast, RNNs allow connections to previous layers and self-connections, resulting in cycles. In the special case of a fully connected recurrent neural network Menguc and Acir [38], the outputs of all neurons are also the inputs of all neurons. The literature is rich with various RNN architectures applied to time series forecasting. Yet, we can generalize the elaboration on various RNNs by stating that they allow having self-connected hidden layers Chen et al. [9]. Compared with feedforward neural networks, RNNs utilize the action of hidden layer *unfolding*, which makes them able to process sequential data. This explains their vast popularity in the analysis of temporal data, such as time series or natural language Cortez et al [14].

A popular RNN architecture is called long short-term memory (LSTM). It was designed by Hochreiter and Schmidhuber [26] to overcome the problems arising when training vanilla RNN models. Traditional RNN training takes a very long time, mostly because of insufficient, decaying error when doing the error backpropagation Guo et al. [23]. The LSTM architecture uses a special type of neurons called memory cells that mimic three kinds of gate operations Hewamalage et al. [25]. These are referred to as the multiplicative input, output, and forget gates. These gates filter out unrelated and perturbed inputs Guo et al. [20]. Standard LSTM models are constructed in a way that past observations influence only the future ones, but there exists a variant called bidirectional LSTM that lifts this restriction Cui et al [15]. Numerous studies show that both unidirectional and bidirectional LSTM networks outperform traditional RNNs due to their ability to capture long-term dependencies more precisely Tang et al. [52].

The Gated Recurrent Unit (GRU) is another RNN model Cho et al. [13]. In comparison with LSTM, GRU executes simplified gate operations by using only two types of memory cells: input merged with output and forget cell Wang et al. [57], called here update and reset gate, respectively Becerra-Rico et al. [6]. As in the case of LSTM, GRU training is less sensitive to the vanishing/exploding gradient problem that is encountered in traditional RNNs Ding et al. [17].

The inclusion of recurrent/delayed connections boosted the capability of neural models to predict time series accurately, while further improvements of their architecture (like LSTM or GRU model) made training dependable. However, it shall be mentioned that the use of the traditional error backpropagation is not the only option to learn network weights from historical data. Alternatively, we can use meta-heuristic approaches to train a model. There exists a range of interesting studies, where the authors used Genetic Algorithm Sadeghi-Niaraki et al. [49] or Ant Colony Optimization ElSaid et al. [19]. The study in Abdulkarim and Engelbrecht [1] concluded that, for the tested time series, dynamic Particle Swarm Optimization obtained a similar forecasting error compared with a feed-forward neural architecture and a recurrent one.

It shall be noted that the application of a modern neural architecture does not relieve a model designer from introducing required data staging techniques. This is why we find a range of domain-dependent studies that link various RNN architectures with supplemental processing options. For example, Liu and Shen [33] used a wavelet transform alongside a GRU architecture, while Nikolaev et al. [43] included a regime-switching step, and Cheng et al. [11] employed wavelet-based de-noising and adaptive neuro-fuzzy inference system.

We should mention recent studies on fusing RNN architectures with Convolutional Neural Networks (CNNs). The latter model has attracted much attention due to its superior efficiency in pattern classification. We find a range of studies [37, 59], where a CNN is merged with an RNN in a deep neural model that aims at time series forecasting. The role of a CNN is to extract features that are used to train an RNN forecasting model Li et al. [32]. Attention mechanisms have also been successfully merged with RNNs, as presented by Zhang et al. [61].

From a high-level perspective on time series forecasting with RNNs, we can also distinguish architectures that read in an entire time series and produce an internal representation of the series, i.e., a network plays the role of an encoder Laubscher [31]. A decoder network then needs to be used to employ this internal representation to produce forecasts Bappy et al. [5]. The described scheme is called an encoder–decoder network and was applied, for example, by Habler and Shabtai [24] together with LSTM, by Chen et al. [10] with convolutional LSTM, and by Yang et al. [60] with GRU.

We shall also mention the forecasting models based on Fuzzy Cognitive Maps (FCMs) Kosko [30]. Such networks are knowledge-oriented architectures with processing capabilities that mimic the ones of RNNs. The most attractive feature of these models is network interpretability. There are numerous papers, including the works of [44, 45, 54] or [58], where FCMs are applied to process temporal data. However, recent studies show that even better forecasting capabilities can be achieved with STCNs Nápoles et al. [39] or long-term cognitive networks Nápoles et al [40]. As far as we know, these FCM generalizations have not yet been used for time series forecasting. This paper extends the research on the STCN model, which will be used as the main building block of our proposal.

3 Short-term cognitive networks

The STCN model was introduced in Nápoles et al. [39] to cope with short-term WHAT-IF simulation problems where problem variables are mapped to neural concepts. In these problems, the goal is to compute the immediate effect of input variables on output ones given long-term prior knowledge. Remark that the model in Nápoles et al. [39] was trained using a gradient-based non-synaptic learning approach devoted to adjusting a set of parametric transfer functions. In this section, we redefine the STCN model such that it can be trained in a synaptic fashion.

The STCN block involves four matrices to perform reasoning: $W_1^{(t)}$, $B_1^{(t)}$, $W_2^{(t)}$, and $B_2^{(t)}$. The first two matrices denote the prior knowledge coming from a previous learning process and can be modified by the experts to include new pieces of knowledge that have not yet been recorded in the historical data (e.g., an expected increase in the Bitcoin value as Tesla decides to accept such cryptocurrency as a valid payment method). These prior knowledge matrices allow for hybrid reasoning, which is an appealing feature of the STCN model. The third and fourth matrices contain learnable weights that adapt the input $X^{(t)}$ and the prior knowledge to the expected output $Y^{(t)}$ in the current step. The matrices $B_1^{(t)}$ and $B_2^{(t)}$ represent the bias weights.

Figure 1 shows how the different components interact with each other in an STCN block. It is important to highlight that this model lacks hidden neurons, so each inner block (abstract layer) has exactly M neurons, with M being the number of neural concepts in the model. This means that we have a neural system in which each component has a well-defined meaning. For example, the intermediate state $H^{(t)}$ represents the outcome that the network would have produced given $X^{(t)}$ if the network would not have been adjusted to the expected output $Y^{(t)}$. Similarly, the bias weights denote the external information that cannot be inferred from the given inputs.

Equations 1 and 2 formalize the short-term reasoning process of this model in the t-th iteration,

$$\hat{Y}^{(t)} = f\left(H^{(t)} W_2^{(t)} \oplus B_2^{(t)}\right) \tag{1}$$

and

$$H^{(t)} = f\left(X^{(t)} W_1^{(t)} \oplus B_1^{(t)}\right) \tag{2}$$

where $X^{(t)}$ and $\hat{Y}^{(t)}$ are $K \times M$ matrices encoding the input and the forecasting in the current iteration, respectively, with K being the number of observations and M the number of neurons. B_1 and B_2 are $1 \times M$ matrices representing the bias weights. $H^{(t)}$ is a $K \times M$ matrix, while $W_1^{(t)}$ and $W_2^{(t)}$ are a $M \times M$ matrices. In these equations, the $\oplus$ operator performs a matrix-vector addition by operating each row of a given matrix with a vector, provided that both the matrix and the vector have the same number of columns. Finally, $f(\cdot)$ stands for the nonlinear transfer function, typically the sigmoid function:

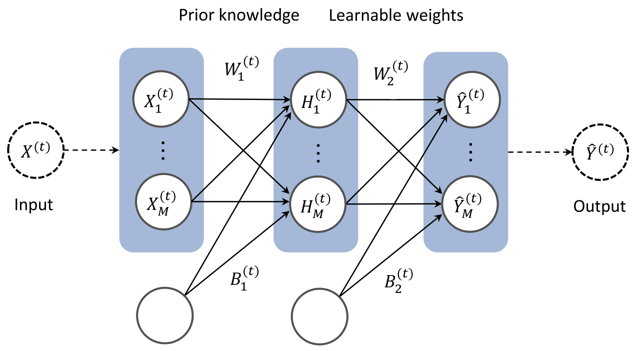

Fig. 1 The STCN block involves two components: the prior knowledge matrices $W_1^{(t)}$ and $B_1^{(t)}$, and the learnable matrices $W_2^{(t)}$ and $B_2^{(t)}$. The prior knowledge matrices are a result of a previous learning process and can be modified by domain experts if deemed opportune

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{3}$$

The inner working of an STCN block can be summarized as follows. The block receives a weight matrix $W_1^{(t)}$, the bias weight matrix $B_1^{(t)}$ and a chunk of data $X^{(t)}$ as the input data. Firstly, we compute an intermediate state $H^{(t)}$ that mixes $X^{(t)}$ with the prior knowledge (e.g., knowledge resulting from the previous iteration). Secondly, we operate $H^{(t)}$ with $W_2^{(t)}$ and $B_2^{(t)}$ to approximate the expected output $Y^{(t)}$.

This short-term reasoning of this model makes it less sensitive to the convergence issues of long-term cognitive networks such as the unique-fixed point attractors Nápoles et al. [39]. Furthermore, the short-term reasoning allows extracting more clear patterns to be used to generate explanations.

4 Long short-term cognitive network

In this section, we introduce the *long short-term cognitive networks* for time series forecasting, which can be defined as a collection of chained STCN blocks.

4.1 Architecture

As mentioned, the model presented in this section is devoted to the multiple-ahead forecast of very long (multivariate) time series. Therefore, the first step is splitting the time series into T time patches, each comprising a collection of tuples with the form $(X^{(t)}, X^{(t+1)})$. In these tuples, the first matrix denotes the input used to feed the network in the current iteration, while the second one is the expected output $Y^{(t)} = X^{(t+1)}$. Notice that each time patch often contains several time steps (e.g., all tuples produced within a 24-hour time frame).

Figure 2 shows, as an example, how to decompose a given time series into T time patches of equal length where each time patch will be processed by an STCN block. This procedure holds for multivariate time series such that both $X^{(t)}$ and $Y^{(t)}$ have a dimension of $K \times M$. In this case, K denotes the number of time steps allocated to the time patch, whereas M defines the width of each STCN block. Therefore, if we have a multivariate time series described by N features and want to forecast L steps, then $M = N \times L$.

In short, the LSTCN model can be defined as a collection of STCN blocks, each processing a specific time patch and passing knowledge to the next block. In each time patch, the matrices of the previous model are aggregated

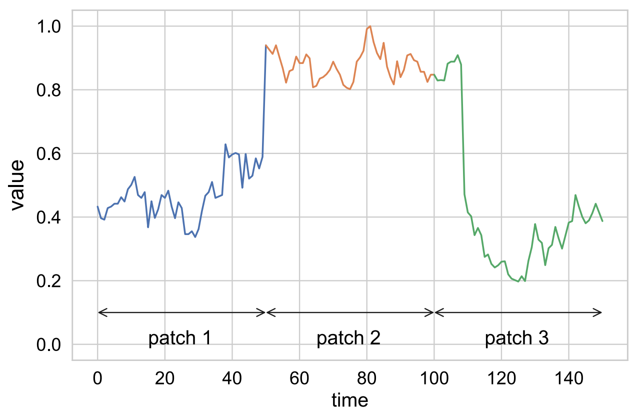

Fig. 2 Recurrent approach to process a (multivariate) time series with an LSTCN model. The sequence is split into T time patches with even length. Each time patch is used to train an STCN block that employs information from the previous block as prior knowledge

and used as prior knowledge for the current STCN block, that is to say:

$$W_1^{(t)} = \Psi\left(W_1^{(t-1)}, W_2^{(t-1)}\right) \tag{4}$$

and

$$B_1^{(t)} = \Psi\left(B_1^{(t-1)}, B_2^{(t-1)}\right) \tag{5}$$

such that $\Psi(x, y) = \tan h(\max\{x, y\})$. The aggregation procedure creates a chained neural structure that allows for long-term predictions since the learned knowledge is used when performing reasoning in the current iteration.

Figure 3 shows the LSTCN architecture to process the time series in Fig. 2, which was split into three time patches of equal length. In the figure, blue boxes represent STCN blocks, while orange boxes denote learning processes.

It should be highlighted that, although the LSTCN model works in a sequential fashion, each STCN block performs an independent learning process (to be explained in the next subsection) before moving to the next block. Therefore, the long-term component refers to how we process the whole sequence, which is done by transferring the knowledge (in the form of weights) from one STCN block to another. Notice that we do not pass the neurons' activation values to the subsequent blocks. Once we have processed the whole sequence, the model narrows down to the last STCN in the pipeline.

We would like to draw attention to a certain design analogy between the LSTCN and the LSTM model. We ought to outline how *short-term* and *long-term* dependencies in temporal data are captured in both models to address this topic. Let us recall that LSTM networks are derived from RNN networks. An RNN network in an unfolded state can be illustrated as a sequence of neural layers. The hidden layers in an RNN are responsible for window-based time series processing. In an RNN, the values computed by the network for the previous time step are used as input when processing the current time step. Due to the cyclic nature of the entire process, training an RNN is challenging. The input signals tend to either decay or grow exponentially. Graves et al. [22] explain that this is referred to in the literature as the vanishing gradient problem. The most significant difference between the RNN and the LSTM model is that the latter adds a forgetting mechanism at each hidden layer. The LSTM model processes the data using a windowing technique in which the number of hidden layers is equal to the length of the window. This window is responsible for processing and recognizing *short-term* dependencies in time series. The forgetting mechanism in each layer acts as a symbolic switch that either retains the incoming signal or forgets it. (Please note that this switch is not binary.) Thus, the forgetting mechanism in LSTM adds flexibility that allows the network to accumulate *long-term* temporal contextual information in its internal states, but at the same time, *short-term* dependencies are also modeled because the processing scheme is still sequential and windows-based.

Similar to the LSTM model, the LSTCN model analyzes data in a sequential, window-based manner (see Fig. 3). The difference is that each STCN block that makes up the LSTCN model can be viewed as a sub-window. The

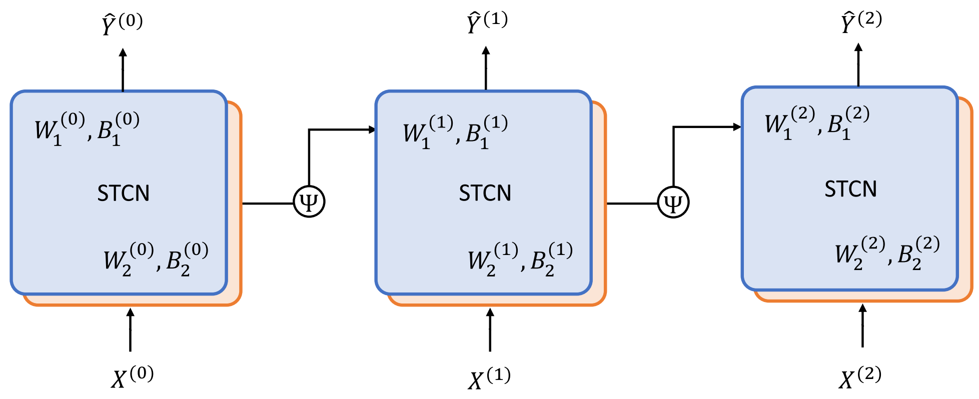

Fig. 3 Example of an LSTCN composed of three STCN blocks. In each iteration, the model receives a time patch $X^{(t)}$ to be processed and produces an approximation of the expected output $Y^{(t)}$. The weights learned in the current block are aggregated (using Eqs. 4 and 5) and transferred to the following STCN block as prior knowledge matrices

aggregation function $\Psi(x, y)$ can roughly be seen as an analogy to the forgetting mechanism in an LSTM. Thus, as a signal is passed through the network, the internal states $H^{(t)}$ of the LSTCN accumulate knowledge of *long-term* temporal contextual information. At the same time, the *short-term* dependencies in the time series are processed in a conventional way, in each STCN block (see Fig. 1).

4.2 Learning

Training an LSTCN model means training each STCN block with its corresponding time patch. In this neural system, the learned knowledge up to the current iteration is stored in B_1 and W_1, while B_2 and W_2 contain the knowledge needed to make the prediction in the current iteration. Therefore, the learning problem consist of computing $W_2^{(t)}$ and $B_2^{(t)}$ given the tuple $(X^{(t)}, Y^{(t)})$ corresponding to the current time patch. Let us recall that $H^{(t)}$ is a $K \times M$ matrix, $W_2^{(t)}$ is a $M \times M$ matrix, while $B_2^{(t)}$ is a $1 \times M$ matrix. The underlying optimization problem is given below:

$$min \rightarrow \left\| f\left(H^{(t)} W_2^{(t)} \oplus B_2^{(t)}\right) - Y^{(t)} \right\|_{\ell_2} + \lambda \left\| \Gamma_2^{(t)} \right\|_{\ell_2} \tag{6}$$

such that

$$\Gamma_2^{(t)} = \begin{bmatrix} W_2^{(t)} \\ B_2^{(t)} \end{bmatrix} \tag{7}$$

represents the matrix with dimension $(K+1) \times M$ that results after performing a row-wise concatenation of the bias weight matrix $B_2^{(t)}$ to $W_2^{(t)}$, while $\lambda \geq 0$ is the ridge regularization penalty. The added value of using a ridge regression approach is regularizing the model and preventing overfitting. In our network, overfitting is likely to happen when splitting the original time series into too many time patches covering few observations.

Equation (8) displays the deterministic learning rule solving this ridge regression problem,

$$\Gamma_2^{(t)} = \left(\left(\Phi^{(t)}\right)^{\top} \Phi^{(t)} + \lambda \Omega^{(t)} \right)^{-1} \left(\Phi^{(t)}\right)^{\top} f^{-}\left(Y^{(t)}\right) \tag{8}$$

where $\Phi^{(t)} = (H^{(t)}|A)$ such that $A_{K \times 1}$ is a column vector filled with ones, $\Omega^{(t)}$ denotes the diagonal matrix of $(\Phi^{(t)})^{\top}\Phi^{(t)}$, while $(\cdot)^{-1}$ represents the Moore–Penrose pseudo-inverse Penrose [46]. This generalized inverse is computed using singular value decomposition and is defined and unique for all real matrices. Remark that this learning rule assumes that the activation values in the inner layer are standardized. As far as standardization is concerned, these calculations are based on standardized activation values. When the final weights are returned, they are adjusted back into their original scale.

It can be noticed that an STCN block trained using the learning rule in Eq. (8) is similar to an Extreme Learning Machine (ELM) Huang et al. [27], which is a special case of a two-layer multilayer perceptron. However, there are three main differences between these models. Firstly, the $W_1^{(t)}$ and $B_1^{(t)}$ matrices are not random but initialized with the prior knowledge arriving at the STCN block from previous learning processes. Secondly, while the hidden layer of ELMs is of arbitrary width, the number of neurons in an STCN is given by the number of steps ahead to be predicted and the number of features in the multivariate time series. Finally, each neuron (also referred to as neural concept) represents the state of a problem feature in a given time step. While this constraint equips our model with interpretability features, it might also limit its approximation capabilities.

Another issue that deserves attention is how to estimate the first weight matrix $W_1^{(0)}$ to be used as prior knowledge in the first iteration. This matrix is expected to be (partially) provided by domain experts or computed from a previous learning process (e.g., using a transfer learning approach). In this paper, we simulate such knowledge by fitting a stateless STCN (that is to say, $H^{(t)} = X^{(t)}$) on a smoothed representation of the whole time series we are processing. The smoothed time series is obtained using the moving average method for a given window size. Finally, we generate some white noise over the computed weights to compensate for the moving average operation. Equation (9) shows how to compute this matrix,

$$W_1^{(0)} \sim \mathcal{N}\left(\left(\bar{X}^{\top}\bar{X} + \lambda\Omega\right)^{-1} \bar{X}^{\top} f^{-}(\bar{Y}), \sigma \right) \tag{9}$$

where $\bar{X}$ and $\bar{Y}$ are the smoothed inputs and outputs obtained for the whole time series, respectively, while σ is the standard deviation. In this case, we will use Ω again to denote the diagonal matrix of $\bar{X}^{\top}\bar{X}$ if no confusion arises.

The prior bias matrix $B_1^{(0)}$ is assumed to be zero since we use that component to model the external stimulus of neurons after performing an STCN's learning process.

The intuition dictates that the training error will go down as more time patches are processed. Of course, such time patches should not be too small to avoid overfitting. In some cases, we might obtain an optimal performance using a single time patch containing the whole sequence such that we will have a single STCN block. In other cases, it might occur that we do not have access to the whole sequence (e.g., as happens when solving online learning problems), such that using a single STCN block would not be an option.

4.3 Interpretability

As mentioned, the architecture of our neural system allows explaining the forecasting since both neurons and weights have a precise meaning for the problem domain being modeled. However, the interpretability cannot be confined to the absence of hidden components in the network since the structure might involve hundreds or thousands of edges.

In this subsection, we introduce a measure to quantify the influence of each feature in the forecasting of *multivariate* time series. Our proposal is based on the knowledge structures of the LSTCN model, i.e., the learned weights connecting the neurons. This implies that our measure is a model-intrinsic feature importance measure, reflecting what the model considers important when learning the relations between the time points. This approach contrasts with model-agnostic methods that inspect how the variations in the input data affect the model's output.

The proposed measure can be computed from $W_1^{(t)}$, $W_2^{(t)}$ or their combination. The scores obtained from $W_1^{(t)}$ can be understood as the feature influence up to the t-th time patch, while scores obtained from $W_2^{(t)}$ can be understood as the feature influence to the current time patch. Let us recall that $W_1^{(t)}$ and $W_2^{(t)}$ are $M \times M$ matrices such that $M = N \times L$, assuming that we have a multivariate time series with N features and that we want to forecast L steps ahead. Moreover, the neurons are organized temporally, which means that we have L blocks of neurons, each containing N units. Equations (10) and (11) show how to quantify the effect of feature f_i on feature f_j given a matrix $W^{(t)}$ that characterizes the interaction among the problem features,

$$\gamma^{(t)}(f_i, f_j) = \sum_{p_i \in P(i)} \sum_{p_j \in P(j)} \left| w_{p_i p_j}^{(t)} \right|, w_{p_i p_j}^{(t)} \in W^{(t)} \tag{10}$$

such that

$$P(i) = \{p \in \mathbb{N}, p \leq M \mid (p \ mod \ i) = 0\}. \tag{11}$$

The *feature influence score* in Equation (10) can be normalized such that the sum of all scores related to the j-th feature is one. This can be done as follows:

$$\hat{\gamma}^{(t)}(f_i, f_j) = \frac{\gamma^{(t)}(f_i, f_j)}{\sum_{k=1}^{N} \gamma^{(t)}(f_k, f_j)}. \tag{12}$$

The rationale behind the proposed feature influence score is that the most important problem features will have attached weights with large absolute values. Moreover, it is expected for the learning algorithm to produce sparse weights with a zero-mean normal distribution, which is an appreciated characteristic when it comes to interpretability.

The idea of computing the relevance of features from the weights in neural systems has been explored in the literature. For example, the Layer-Wise Relevance Propagation (LRP) algorithm Bach et al. [4] explains the predictions made by a neural classifier for a given instance by assigning relevance scores to features, which are computed using the learned weights and neurons' activation values. It should be stated that we do not use neurons' activation values in our feature influence score as we intend to produce global explanations based on the learned weights only. Similar approaches have been proposed in Nápoles et al. [41] and Nápoles et al. [42] but applied to LTCN-based classifiers. In the first study, the feature scores indicate which features play a significant role in obtaining a given class instead of an alternative class. This type of interpretability responds to the question *why not?*. Conversely, the second study measures the feature importance in obtaining the decision class. The results were contrasted with the feature scores obtained from logistic regression and both models agreed on the top features that play a role in the outcome. Both feature score measures operate on neural systems where the neurons have an explicit meaning for the problem domain. Therefore, the learned weights can be used as a proxy for interpretability.

5 Numerical simulations

In this section, we will explore the performance (forecasting error and training time) of our neural system on three case studies involving univariate and multivariate time series. In the case of multivariate time series, we will also depict the feature contribution score to explain the predictions.

When it comes to the pre-processing steps, we interpolate the missing values (whenever applicable) and normalize the series using the *min-max* method. In addition, we split the series into 80% for training and validation and 20% for testing purposes. As for the performance metric, we use the *mean absolute error* in all simulations reported in the section. For the sake of convenience, we trimmed the training sequence (by deleting the first observations) such that the number of times is a multiple of L (the number of steps ahead we want to forecast).

The models used for comparison are a fully connected Recurrent Neural Network (RNN) where the output is to be fed back to the input, GRU, LSTM and Extreme Learning Machine (ELM). In the first three models, the number of epochs was set to 20, while the batch size was obtained through hyperparameter tuning (using grid search). The candidate batch sizes were the powers of two, starting from 32 until 4,096. The values for the remaining parameters were retained as provided in the Keras library. In the case of the LSTCN model, we fine-tuned the number of time patches $T \in \{1, 2..., 10\}$ and the regularization penalty $\lambda \in \{1.0E{-}{-}3,\ 1.0E{-}{-}2,\ 1.0E{-}{-}1, 1.0E{+}1, 1.0E{+}2, 1.0E{+}3\}$. In Eq. (9), we arbitrarily set the standard deviation σ to 0.05 and the moving window size w to 100. These two hyperparameters were not optimized during the hyperparameter tuning step as they were used to simulate the prior knowledge component. In the case of ELM, we use the implementation provided in Scikit–ELM [3]. The number of neurons in the hidden layer was set to $M = N \times L$ where N is the number of features while L is the number of steps ahead to be forecast. The values for the remaining hyperparameters were retained as provided in the library.

Finally, all experiments presented in this section were performed on a high-performance computing environment that uses two Intel Xeon Gold 6152 CPUs at 2.10 GHz, each with 22 cores and 768 GB of memory.

5.1 Apple Health's step counter

The first case study concerns physical activity prediction based on daily step counts. In this case study, the health data of one individual were extracted from the Apple Health application in the period from 2015 to 2021. In total, the time series dataset is composed of 79,142 instances or time steps. The Apple Health application records the step counts in small sessions during which the walking occurs. The dataset (available at https://bit.ly/2S9vzMD) contains two timestamps (start date and end date), the number of recorded steps, and separate columns for year, month, date, day, and hour. Besides, the day of the week that each value was recorded is known. Table 1 presents descriptive statistics attached to this univariate time series before normalization.

The target variable (number of steps) follows an exponential distribution with very infrequent, extremely high step counts and very common low step counts. Overall, the data neither follows seasonal patterns nor a trend.

Table 2 shows the normalized errors attached to the models under consideration when forecasting 50 steps ahead in the Steps dataset. In addition, we portray the training and test times (in seconds) for the optimized models. The hyperparameter tuning reported that our neural system needed two iterations to produce the lowest forecasting errors, while the optimal batch size for RNN, GRU and LSTM was found to be 32.

Table 1 Descriptive statistics for the steps case study

Variable	Mean	Std	Min	Max
Value	191.63	235.73	1.00	7,205.00

Table 2 Simulation results for the steps case study

Method	Error		Time	
	Training	Test	Training	Test
RNN	0.0241	0.0249	5.513	0.444
GRU	0.0236	0.0251	46.002	0.703
LSTM	0.0237	0.0248	44.689	0.933
ELM	0.0234	0.0245	0.010	0.00
LSTCN	0.0221	0.0212	0.0202	0.001

Although LSTCN outperforms the remaining methods in terms of forecasting error, what is truly remarkable is its efficiency. The results show that LSTCN is 2.7E+2 times faster than RNN, 2.3E+3 times faster than GRU, 2.2E+3 times faster than LSTM, and just 2.0E-2 times slower than ELM. In this experiment, we ran all models five times with optimized hyperparameters and selected the shortest training time in each case. Hence, the time measures reported in Table 2 concern the fastest executions observed in our simulations.

Figure 4 displays the distributions of weights in the W_1 and W_2 matrices attached to the last STCN block (the one to be used to perform the forecasting). In other words, we visualize the differences in the distributions of prior knowledge weights and the weights learned in the last STCN block. It is worth recalling that the prior knowledge block in that last block is what the network has learned after processing all the time patches but the last one. In contrast, the learned weights in that block adapt the prior knowledge to forecast the last time patch. In this case study, most prior knowledge weights are distributed in the $[-0.2, 1.0]$ interval, while weights in W_2 follows a zero-mean Gaussian distribution. This figure illustrates that the network significantly adapts the prior knowledge to the last piece of data available.

Figure 5 depicts the overall behavior of weights connecting the inner neurons with the outer ones in the last

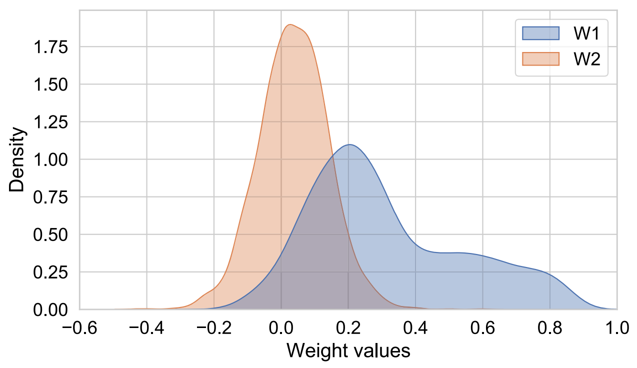

Fig. 4 Distribution of weights for the Steps case study

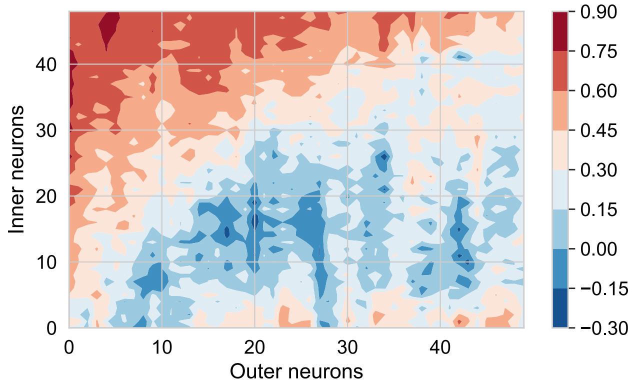

Fig. 5 Behavior of weights connecting the inner neurons with the outer ones in the last STCN block after averaging W_1 and W_2

STCN block. In this simulation, we averaged the W_1 and W_2 matrices for the sake of simplicity, thus resulting in an average layer. In that layer, inner and outer neurons refer to the leftmost and rightmost neurons, respectively. Observe that the learning algorithm assigns larger weights to connections between neurons processing the last steps in the input sequence and neurons processing the first steps in the output sequence. This is an expected behavior in time series forecasting that supports the rationale of the proposed feature relevance measure.

The fact that each neuron has a well-defined meaning for the problem domain makes it possible to elucidate how the network uses the current L time steps to predict the following ones. Using that knowledge, experts could estimate how many previous time steps would be needed to predict a sequence of length L without performing further simulations.

5.2 Household electric power consumption

The second case study concerns the energy consumption in one house in France measured each minute from December 2006 to November 2010 (47 months). This dataset (available at https://bit.ly/3ugv8Pt) involves nine features and 2,075,259 observations from which 1.25% are missing. Hence, records with missing values were interpolated using the nearest neighbor method. In our experiments, we retained the following variables: global minute-averaged active power (in kilowatt), global minute-averaged reactive power (in kilowatt), minute-averaged voltage (in volt), and global minute-averaged current intensity (in ampere) (Table 3).

The series exhibits cyclic patterns. On the most fine-grained scale, we observe a repeating low nighttime power consumption. We also noted a less distinct but still present pattern related to the day of the week: higher power consumption during weekend days. Finally, we observed high power consumption during the winter months (peaks in January) each year and low in summer (lowest values recorded for July).

Table 4 portrays the normalized errors obtained by each optimized model when forecasting 200 steps ahead, and the training and test times (in seconds). The hyperparameter tuning reported that our network produced the optimal forecasts with two iterations, while the optimal batch size for RNN, GRU and LSTM was 4,096, 64 and 256, respectively. According to these simulations, LSTCN obtains the best results followed by GRU with the latter being notably slower than the former. Overall, LSTCN proved to be 2.3E+1 times faster than RNN, 2.2E+3 times faster than GRU, and 1.6E+3 times faster than LSTM. ELM was the second-fastest algorithm and showed competitive results in terms of error compared to LSTCN.

Figure 6 displays the distributions of weights in the W_1 and W_2 matrices for the last STCN block. These histograms reveal that weights follow a zero-mean Gaussian distribution and that the second matrix has more weights near zero (the shape of the second curve contracts toward zero). In this case study, the network does not shift the distribution of weights as happened in the first case study. Actually, the accumulated prior knowledge does not seem to suffer much distortion (distribution-wise) when adapted to the last time patch.

Figure 7 displays the feature influence scores obtained with Equation (12). These scores were computed after averaging the W_1 and W_2 matrices that result from adjusting the network to the last time patch. In this figure, the bubble size denotes the extent to which one feature in

Table 3 Descriptive statistics concerning four variables in the Power case study: mean value, standard deviation, minimal, and maximal values (“pwr” stands for power)

Variable	Mean	Std	Min	Max
Active pwr	1.09	1.06	0.08	11.12
Reactive pwr	0.12	0.11	0.00	1.39
Voltage	240.84	3.24	223.20	254.15
Intensity	4.63	4.44	0.20	48.40

Table 4 Simulation results for the power case study

Method	Error		Time	
	Training	Test	Training	Test
RNN	0.3326	0.3359	14.56	0.76
GRU	0.0581	0.0547	1373.58	2.21
LSTM	0.0637	0.0581	991.51	2.81
ELM	0.054	0.0581	0.781	0.079
LSTCN	0.0559	0.0531	0.63	0.04

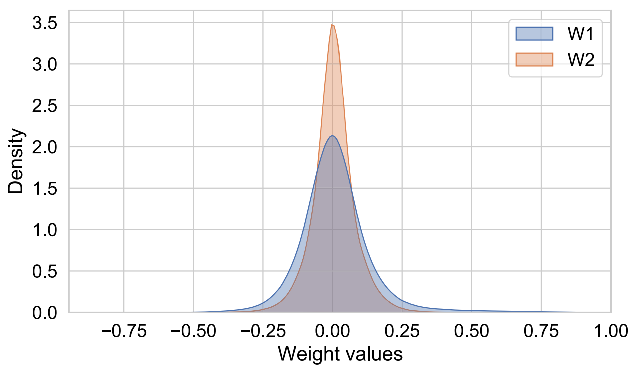

Fig. 6 Distribution of weights for the Power case study

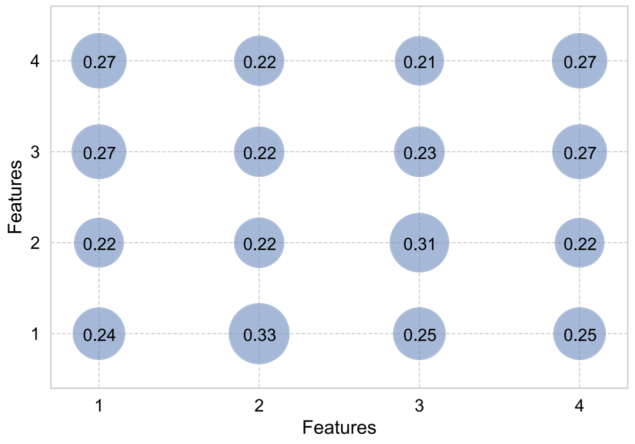

Fig. 7 Feature influence in the Power case study

the y-axis is deemed relevant to forecast the value of another feature in the x-axis. For example, it was observed that the first feature (global active power) is the most important one to forecast the second feature (global reactive power). Observe that the sum of all scores by column is one due to the normalization step.

Overall, the results indicate that the proposed network obtains small forecasting errors while being markedly faster than the state-of-the-art recurrent neural networks. Moreover, its knowledge structures facilitate explaining how the forecasting was made, using feature relevance explanations.

5.3 Bitcoin transactions analysis

In this section, we inspect a case study concerning changes in the Bitcoin transaction graph observed with a daily frequency from January 2009 to December 2018. The data set is publicly available in the UCI Repository (https://bit.ly/3ES71M1). Using a time interval of 24 hours, the contributors of this dataset Akcora et al. [2] extracted daily transactions and characterized them. In total, we have

Table 5 Descriptive statistics (mean, standard deviation, minimum and maximum value) of variables in the Bitcoin dataset

Variable	Mean	Std	Min	Max
Length	45.01	58.98	0.00	144.00
Weight	0.55	3.67	0.00	1,943.75
Count	721.64	1,689.68	1.00	14,497.00
Looped	238.51	966.32	0.00	14,496.00
Neighbors	2.21	17.92	1.00	12,920.00
Income	4.47e+09	1.63e+11	3.00e+07	5.00e+13

Table 6 Simulation results for the Bitcoin case study

Method	Error		Time	
	Training	Test	Training	Test
RNN	0.3003	0.3146	32.81	1.56
GRU	0.0653	0.0918	2596.92	5.07
LSTM	0.0664	0.0872	3488.91	6.96
ELM	0.0591	0.1	2.2199	0.254
LSTCN	0.0583	0.0774	1.56	0.09

2,916,697 observations of six numerical features (the remaining ones were discarded).

Due to the nature of this dataset, we do not observe typical statistical properties (there are no seasonal patterns, the data are not stationary, the fluctuations do not show evident patterns and features are not normally distributed). Table 5 depicts descriptive statistics for the retained features.

Table 6 shows the errors obtained by each optimized network when forecasting 200 steps ahead, and the training and test times (in seconds). After performing hyperparameter tuning, we found that the optimal batch size for RNN, GRU and LSTM was 4,096, 128 and 64, respectively, while the number of LSTCN iterations was set to

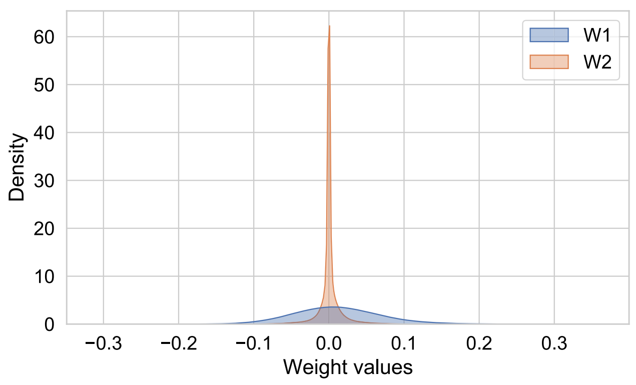

Fig. 8 Distribution of weights in the Bitcoin case study

eight. In this problem, the LSTCN model clearly outperformed the remaining algorithms selected for comparison. When it comes to the training time, LSTCN is 2.1E+1 times faster than RNN, 1.7E+3 times faster than GRU, 2.2E+3 times faster than LSTM, and 1.4 times faster than ELM. Similarly to the other experiments, ELM was the second-fastest algorithm; however, it obtained the second-worst score in terms of the test error.

Figure 8 shows the distribution of weights in the first prior knowledge matrix and the matrix computed in the last learning process. This figure illustrates how the weights become more sparse as the network performs more iterations. This happens due to a heavy ℓ_2 regularization with $\lambda = 1.0E + 3$ being the best penalty value obtained with grid search. However, no shift in the distributions is observed.

Figure 9 shows the feature influence scores obtained for the Bitcoin case study. Similarly to the previous scenario, these scores were computed after averaging the W_1 and W_2 matrices that result from adjusting the network to the last time patch. The relevance scores suggest that the sixth (income), the second (weight) and the third (count) features have the biggest influence in the forecasting.

Overall, it should be highlighted that the intention of the proposed feature score is to provide the LSTCN with intrinsic interpretability, as opposed to model-agnostic measures of feature importance. In general, model-intrinsic explanations are preferred when the fidelity of the explanations to the model is an important factor for the user Rudin [48]. In practice, our approach can help the practitioners elucidate the degree to which each feature influences the forecasting. Comparing the quality of our model-intrinsic explanations with model-agnostic explanations would require specific application and the availability of experts to measure satisfaction, informativeness, usefulness, trust, etc. Doshi-Velez and Kim [18]. This is an interesting step to take into account in the future work of this research line.

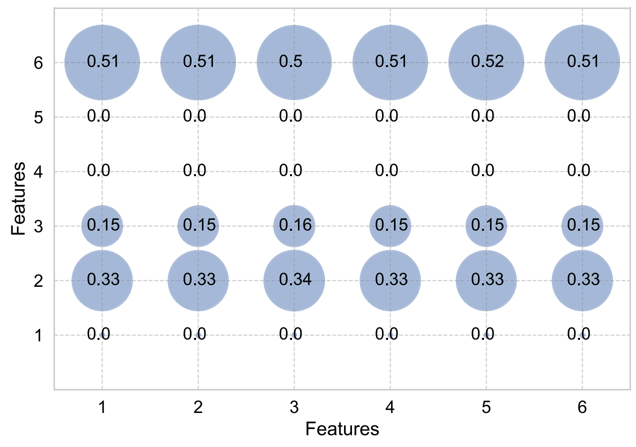

Fig. 9 Feature influence in the Bitcoin case study

6 Concluding remarks

In this paper, we have presented a recurrent neural system termed Long Short-term Cognitive Networks to forecast long time series. The proposed model consists of a collection of STCN blocks, each processing a specific data chunk (time patch). In this neural ensemble, each STCN block passes information to the next one in the form of prior knowledge matrices that preserve what the model has learned up to the current iteration. This means that, in each iteration, the learning problem narrows down to solving a regression problem. Furthermore, neurons and weights can be mapped to the problem domain, making our neural system interpretable.

The underlying model aims at solving an optimization problem, which is practically realized with ridge regression. The natural limitation of such a solution is that we may encounter computational issues in ultra-high dimensional spaces. The literature of the domain suggests solving these issues with the help of dimensionality reduction algorithms (see [12, 55, 56]).

The numerical simulations using three case studies allow us to draw the following conclusions. Firstly, our model performs better than (or comparably to) state-of-the-art recurrent neural networks. It has not escaped our notice that these algorithms could have produced smaller forecasting errors if we had optimized other hyperparameters (such as the learning rate, the optimizer, the regularizer, etc.). However, such an increase in performance would come at the expense of a significant increase in the computations needed to produce fully optimized models. Secondly, the simulation results have shown that our proposal is noticeably faster than GRU and LSTM, which are popular recurrent models for time series forecasting, and comparable to ELM. Such a conclusion is particularly relevant since our primary goal was to design a fairly accurate forecasting model with fast training time rather than outperforming the forecasting capabilities of these recurrent models. Finally, we have illustrated how to derive insights into the relevance of features using the network's knowledge structures with little effort.

Future research efforts will be devoted to exploring the forecasting capabilities of our model further. On the one hand, we plan to conduct a larger experiment involving more univariate and multivariate time series. On the other hand, we will analytically study the generalization properties of LSTCNs under the PAC-Learning formalism. This seems especially interesting since the network's size depends on the number of features and the number of steps ahead to be forecast.

Funding Y. Salgueiro would like to acknowledge the support provided by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID and the super-computing infrastructure of the NLHPC (ECM-02).

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Abdulkarim SA, Engelbrecht AP (2019) Time series forecasting using neural networks: are recurrent connections necessary? Neural Process Lett 50(3):2763–2795. https://doi.org/10.1007/s11063-019-10061-5
2. Akcora CG, Li Y, Gel YR, et al (2020) Bitcoinheist: topological data analysis for ransomware prediction on the bitcoin blockchain. In: Bessiere C (ed) Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20. International joint conferences on artificial intelligence organization, pp 4439–4445. https://doi.org/10.24963/ijcai.2020/612
3. Akusok A, Leal LE, Björk KM, et al (2021) Scikit-ELM: an extreme learning machine toolbox for dynamic and scalable learning. In: Proceedings of the 2019 international conference on extreme learning machine. Springer, pp 69–78
4. Bach S, Binder A, Montavon G et al (2015) On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS ONE 10(7):e0130140. https://doi.org/10.1371/journal.pone.0130140
5. Bappy JH, Simons C, Nataraj L et al (2019) Hybrid lstm and encoder-decoder architecture for detection of image forgeries. IEEE Trans Image Process 28(7):3286–3300. https://doi.org/10.1109/TIP.2019.2895466
6. Becerra-Rico J, Aceves-Fernández MA, Esquivel-Escalante K et al (2020) Airborne particle pollution predictive model using gated recurrent unit (GRU) deep neural networks. Earth Sci Inf 13(3):821–834. https://doi.org/10.1007/s12145-020-00462-9
7. Bhaskar K, Singh SN (2012) AWNN-assisted wind power forecasting using feed-forward neural network. IEEE Trans Sustain Energy 3(2):306–315. https://doi.org/10.1109/TSTE.2011.2182215
8. Box GE, Jenkins GM, Reinsel GC et al (2015) Time series analysis: forecasting and control. Wiley, New York
9. Chen J, Jing H, Chang Y et al (2019) Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process. Reliab Eng Syst Saf 185:372–382. https://doi.org/10.1016/j.ress.2019.01.006
10. Chen K, Song X, Han D et al (2020) Pedestrian behavior prediction model with a convolutional LSTM encoder-decoder. Physica A 560(125):132. https://doi.org/10.1016/j.physa.2020.125132
11. Cheng L, Zang H, Ding T et al (2018) Ensemble recurrent neural network based probabilistic wind speed forecasting approach. Energies. https://doi.org/10.3390/en11081958
12. Cho H, Fryzlewicz P (2012) Multiscale and multilevel technique for consistent segmentation of nonstationary time series. Stat Sin 22(1):207–229. https://doi.org/10.5705/ss.2009.280
13. Cho K, van Merriënboer B, Gulcehre C, et al (2014) 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). Association for Computational Linguistics, Doha, Qatar, pp 1724–1734. https://doi.org/10.3115/v1/D14-1179
14. Cortez B, Carrera B, Kim YJ et al (2018) An architecture for emergency event prediction using LSTM recurrent neural networks. Expert Syst Appl 97:315–324. https://doi.org/10.1016/j.eswa.2017.12.037
15. Cui Z, Ke R, Pu Z et al (2020) Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting network-wide traffic state with missing values. Transp Res Part C Emerg Technol 118(102):674. https://doi.org/10.1016/j.trc.2020.102674
16. Devlin J, Chang MW, Lee K, et al (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:181004805
17. Ding M, Zhou H, Xie H et al (2019) A gated recurrent unit neural networks based wind speed error correction model for short-term wind power forecasting. Neurocomputing 365:54–61. https://doi.org/10.1016/j.neucom.2019.07.058
18. Doshi-Velez F, Kim B (2018) Considerations for evaluation and generalization in interpretable machine learning. In: Explainable and interpretable models in computer vision and machine learning. Springer, Berlin, pp 3–17
19. ElSaid A, El Jamiy F, Higgins J et al (2018) Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. Appl Soft Comput 73:969–991. https://doi.org/10.1016/j.asoc.2018.09.013
20. Gao X, Shi M, Song X et al (2019) Recurrent neural networks for real-time prediction of TBM operating parameters. Autom Constr 98:225–235. https://doi.org/10.1016/j.autcon.2018.11.013
21. Grau I, Sengupta D, Lorenzo MMG, et al (2020) An interpretable semi-supervised classifier using rough sets for amended self-labeling. In: IEEE international conference on fuzzy systems (FUZZ-IEEE), pp 1–8. https://doi.org/10.1109/FUZZ48607.2020.9177549
22. Graves A, Liwicki M, Fernández S et al (2009) A novel connectionist system for unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 31(5):855–868. https://doi.org/10.1109/TPAMI.2008.137
23. Guo L, Li N, Jia F et al (2017) A recurrent neural network based health indicator for remaining useful life prediction of bearings. Neurocomputing 240:98–109. https://doi.org/10.1016/j.neucom.2017.02.045
24. Habler E, Shabtai A (2018) Using LSTM encoder-decoder algorithm for detecting anomalous ADS-B messages. Comput Secur 78:155–173. https://doi.org/10.1016/j.cose.2018.07.004
25. Hewamalage H, Bergmeir C, Bandara K (2021) Recurrent neural networks for time series forecasting: current status and future directions. Int J Forecast 37(1):388–427. https://doi.org/10.1016/j.ijforecast.2020.06.008
26. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

27. Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1):489–501. https://doi.org/10.1016/j.neucom.2005.12.126
28. Hyndman R, Koehler AB, Ord JK et al (2008) Forecasting with exponential smoothing: the state space approach. Springer, Berlin
29. Kong W, Dong ZY, Jia Y et al (2019) Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Trans Smart Grid 10(1):841–851. https://doi.org/10.1109/TSG.2017.2753802
30. Kosko B (1986) Fuzzy cognitive maps. Int J Man Mach Stud 24(1):65–75. https://doi.org/10.1016/S0020-7373(86)80040-2
31. Laubscher R (2019) Time-series forecasting of coal-fired power plant reheater metal temperatures using encoder-decoder recurrent neural networks. Energy 189(116):187. https://doi.org/10.1016/j.energy.2019.116187
32. Li K, Daniels J, Liu C et al (2020) Convolutional recurrent neural networks for glucose prediction. IEEE J Biomed Health Inform 24(2):603–613. https://doi.org/10.1109/JBHI.2019.2908488
33. Liu H, Shen L (2020) Forecasting carbon price using empirical wavelet transform and gated recurrent unit neural network. Carbon Manag 11(1):25–37. https://doi.org/10.1080/17583004.2019.1686930
34. Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. In: Guyon I, Luxburg UV, Bengio S et al (eds) Advances in neural information processing systems, vol 30. Curran Associates Inc, New York, pp 4765–4774
35. Makridakis S, Spiliotis E, Assimakopoulos V (2018) The m4 competition: results, findings, conclusion and way forward. Int J Forecast 34(4):802–808. https://doi.org/10.1016/j.ijforecast.2018.06.001
36. Makridakis S, Spiliotis E, Assimakopoulos V (2018b) Statistical and machine learning forecasting methods: Concerns and ways forward. PloS one 13(3):e0194889. https://doi.org/10.1371/journal.pone.0194889
37. Mallick T, Balaprakash P, Rask E et al (2020) Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. Transp Res Rec 2674(9):473–488. https://doi.org/10.1177/0361198120930010
38. Menguc EC, Acir N (2018) Kurtosis-based CRTRL algorithms for fully connected recurrent neural networks. IEEE Trans Neural Netw Learn Syst 29(12):6123–6131. https://doi.org/10.1109/TNNLS.2018.2826442
39. Nápoles G, Vanhoenshoven F, Vanhoof K (2019) Short-term cognitive networks, flexible reasoning and nonsynaptic learning. Neural Netw 115:72–81. https://doi.org/10.1016/j.neunet.2019.03.012
40. Nápoles G, Vanhoenshoven F, Falcon R et al (2020) Nonsynaptic error backpropagation in long-term cognitive networks. IEEE Trans Neural Netw Learn Syst 31(3):865–875. https://doi.org/10.1109/TNNLS.2019.2910555
41. Nápoles G, Jastrzebska A, Salgueiro Y (2021) Pattern classification with evolving long-term cognitive networks. Inf Sci 548:461–478. https://doi.org/10.1016/j.ins.2020.08.058
42. Nápoles G, Salgueiro Y, Grau I, et al (2021) Recurrence-aware long-term cognitive network for explainable pattern classification. IEEE Trans Cybern. arXiv:abs/2107.03423
43. Nikolaev NY, Smirnov E, Stamate D et al (2019) A regime-switching recurrent neural network model applied to wind time series. Appl Soft Comput 80:723–734. https://doi.org/10.1016/j.asoc.2019.04.009
44. Papageorgiou EI, Poczeta K (2017) A two-stage model for time series prediction based on fuzzy cognitive maps and neural networks. Neurocomputing 232:113–121. https://doi.org/10.1016/j.neucom.2016.10.072
45. Pedrycz W, Jastrzebska A, Homenda W (2016) Design of fuzzy cognitive maps for modeling time series. Trans Fuz Sys 24(1):120–130. https://doi.org/10.1109/TFUZZ.2015.2428717
46. Penrose R (1955) A generalized inverse for matrices. Math Proc Camb Philos Soc 51(3):406–413. https://doi.org/10.1017/S0305004100030401
47. Ribeiro MT, Singh S, Guestrin C (2016) Why should i trust you?: explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery, New York, pp 1135–1144. https://doi.org/10.1145/2939672.2939778
48. Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215. https://doi.org/10.1038/s42256-019-0048-x
49. Sadeghi-Niaraki A, Mirshafiei P, Shakeri M et al (2020) Short-term traffic flow prediction using the modified elman recurrent neural network optimized through a genetic algorithm. IEEE Access 8:217526–217540. https://doi.org/10.1109/ACCESS.2020.3039410
50. Smyl S (2020) A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. Int J Forecast 36(1):75–85. https://doi.org/10.1016/j.ijforecast.2019.03.017
51. Strubell E, Ganesh A, McCallum A (2020) Energy and policy considerations for modern deep learning research. Proc AAAI Conf Artif Intell 34(09):13693–13696. https://doi.org/10.1609/aaai.v34i09.7123
52. Tang X, Dai Y, Wang T et al (2019) Short-term power load forecasting based on multi-layer bidirectional recurrent neural network. IET Gener Transm Distrib 13(17):3847–3854. https://doi.org/10.1049/iet-gtd.2018.6687
53. Ticknor JL (2013) A Bayesian regularized artificial neural network for stock market forecasting. Expert Syst Appl 40(14):5501–5506. https://doi.org/10.1016/j.eswa.2013.04.013
54. Vanhoenshoven F, Nápoles G, Froelich W et al (2020) Pseudoinverse learning of fuzzy cognitive maps for multivariate time series forecasting. Appl Soft Comput 95(106):461. https://doi.org/10.1016/j.asoc.2020.106461
55. Wang H (2012) Factor profiled sure independence screening. Biometrika 99(1):15–28. https://doi.org/10.1093/biomet/asr074
56. Wang X, Leng C (2016) High dimensional ordinary least squares projection for screening variables. J R Stat Soc Ser B (Stat Methodol) 78(3):589–611. https://doi.org/10.1111/rssb.12127
57. Wang Y, Liu M, Bao Z et al (2018) Short-term load forecasting with multi-source data using gated recurrent unit neural networks. Energies. https://doi.org/10.3390/en11051138
58. Wu K, Liu J (2017) Learning large-scale fuzzy cognitive maps based on compressed sensing and application in reconstructing gene regulatory networks. IEEE Trans Fuzzy Syst 25(6):1546–1560. https://doi.org/10.1109/TFUZZ.2017.2741444
59. Xue X, Feng J, Gao Y et al (2019) Convolutional recurrent neural networks with a self-attention mechanism for personnel performance prediction. Entropy. https://doi.org/10.3390/e21121227
60. Yang Z, Liu Z, Lu Y et al (2021) Multi-indices quantification for left ventricle via densenet and GRU-based encoder-decoder with attention. Complexity 3260:259. https://doi.org/10.1155/2021/3260259
61. Zhang M, Yu Z, Xu Z (2020) Short-term load forecasting using recurrent neural networks with input attention mechanism and hidden connection mechanism. IEEE Access 8:186514–186529. https://doi.org/10.1109/ACCESS.2020.3029224