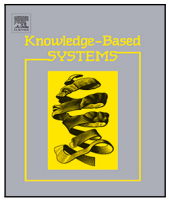
Backpropagation through time learning for recurrence-aware long-term cognitive networks

Knowledge-Based Systems 295 (2024) 111825

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Backpropagation through time learning for recurrence-aware long-term cognitive networks

Gonzalo Nápoles [a,*], Agnieszka Jastrzebska [b], Isel Grau [c], Yamisleydi Salgueiro [d]

[a] Department of Cognitive Science & Artificial Intelligence, Tilburg University, The Netherlands
[b] Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
[c] Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, The Netherlands
[d] Department of Industrial Engineering, Faculty of Engineering, Universidad de Talca, Campus Curicó, Chile

ARTICLE INFO

Keywords:
Fuzzy cognitive maps
Recurrence-aware long-term cognitive network
Backpropagation through time

ABSTRACT

Fuzzy Cognitive Mapping (FCM) and the extensive family of models derived from it have firmly established their strong position in the landscape of machine learning algorithms. Specifically designed for pattern classification and multi-output regression, the recently introduced Recurrence-aware Long-term Cognitive Network (r-LTCN) model is one of these FCM-inspired extensions. On the one hand, this recurrent neural network connects all temporal states generated during the reasoning process with the decision-making layer. On the other hand, it uses a quasi-nonlinear reasoning rule devoted to avoiding convergence issues caused by unique fixed points, which typically emerge in other FCM models. In the original paper, the authors employed a combination of unsupervised and supervised learning to compute the r-LTCNs' learnable parameters. Despite r-LTCNs' astounding performance for a wide variety of pattern classification problems, the literature reports no attempt to train these recurrent neural systems in a fully supervised manner nor provide insights into their performance in other machine learning settings. This paper brings forward a modified Backpropagation Through Time learning (BPTT) algorithm devoted to training r-LTCN models used for multi-output regressions tasks rather than pattern classification. The proposed BPPT includes a simple yet effective mechanism to deal with the vanishing gradient within the recurrent layer that operates as a closed system while being tailored to the quasi-nonlinear reasoning mechanism. Empirical evaluation of the proposed BPTT algorithm using 20 multi-output regression problems reveals that it produces lower prediction errors compared with other state-of-the-art learning approaches.

1. Introduction

Prediction is a ubiquitous subject within the realm of machine learning, encompassing various popular approaches such as linear regression, decision trees, support vector machines, neural networks, and ensemble methods like random forests and gradient boosting. These predictive models play a crucial role in facilitating intricate decision-making processes for individuals and organizations alike. Furthermore, predictions underpin a myriad of automated procedures, serving as the cornerstone for modern operational frameworks and driving efficiency across diverse sectors and application domains.

Fuzzy Cognitive Maps (FCMs) [1] are knowledge-based recurrent neural networks that hold promise in tackling prediction tasks spanning regression, pattern classification, and time series forecasting. Contrary to other prediction methods, FCMs offer a distinctive advantage: domain experts can infuse the model with knowledge, allowing a degree of hybrid intelligence. Additionally, FCMs boast interpretability features since neurons and weights connecting them encapsulate a well-defined meaning relevant to the modeled problem.

Despite being successful in several applications, the drawback of FCM models lies in their limited approximation capabilities. Such a longstanding issue is motivated by four primary factors. Firstly, the size of these cognitive networks is ultimately conditioned by the number of problem variables. Secondly, both the neurons' activation values and causal weights are confined to a strict interval. Thirdly, FCM models often suffer from convergence issues, leading to poor approximations. Finally, there is still a lack of mathematically-solid learning algorithms for computing the learnable parameters.

Aiming to enhance the approximation capabilities of cognitive mapping, the authors in [2] proposed the Long-term Cognitive Network (LTCN) model. From a technical viewpoint, the primary distinction between FCMs and LTCNs lies in the assumed weight ranges. While

* Corresponding author.
E-mail address: g.r.napoles@uvt.nl (G. Nápoles).

https://doi.org/10.1016/j.knosys.2024.111825
Received 28 September 2023; Received in revised form 31 March 2024; Accepted 16 April 2024
Available online 26 April 2024

the FCM model confines weights to the $[-1, 1]$ domain, the LTCN model imposes no such restrictions. However, this difference might be considered marginal since FCM and LTCN are governed by the same reasoning rule, where the number of learnable parameters does not depend on the number of iterations. Moreover, both models are still susceptible to undesirable convergence issues, such as the presence of unique fixed-point attractors [3].

The Recurrence-aware Long-term Cognitive Networks (r-LTCNs) [4] improved the LTCN model by introducing two algorithmic modifications. On the one hand, r-LTCNs employ a parameterized quasi-nonlinear reasoning rule [5], which is different in both conceptual design and operation from the traditional FCM reasoning rule. As a result, the hidden states of the network in the current iteration are not given solely by the states in the previous iteration but also by the initial conditions. The amount of information coming from the initial conditions and the network's state in the previous iteration is regulated by the quasi-nonlinear reasoning rule and a dedicated parameter. On the other hand, in contrast to FCMs or LTCNs, where the decisions are not made from the network's state in the last iteration, the r-LTCN model connects all hidden states with the decision-making layer. Consequently, the number of learnable parameters increases with the number of iterations, enhancing the model's approximation capabilities.

In the literature, only one method devoted to training r-LTCNs has been reported [4]. It employed unsupervised learning to compute the inner weights (connecting the input neurons between them) and a supervised learning rule to compute the outer weights (connecting the input and output neurons). To the best of our knowledge, no other methods suitable to train r-LTCN models using quasi-nonlinear reasoning in a fully supervised way have been reported. Moreover, although the r-LTCN model clearly outperformed other pure FCM classifiers in terms of prediction rates [6], its performance has only been tested in pattern classification problems.

In this paper, we propose a revised Backpropagation Through Time (BPTT) algorithm for training r-LTCNs in a fully supervised manner. While the BPTT paradigm is the natural choice when training recurrent neural systems, its usage has been neither mainstream nor effective when it comes to pure FCM models. The main reasons behind this unfortunate outcome rely on the inherent difficulties caused by the unique fixed-point attractors, the inner working of FCMs as closed systems that do not receive any fresh input in each iteration, and the vanishing gradient problem [7,8]. It is worth mentioning that in the r-LTCN's architecture, the gradient is likely to vanish only within the recurrent reasoning block (running the inner weight matrix). However, it does not imply that the network stops learning, as it might happen with other recurrent systems. For example, outer weights (present in the recurrence-aware connections) might become more important for the network's decision-making since the gradients associated with outer weights are unlikely to vanish. This happens because the recurrence-aware mechanism brings the error signal obtained in the last iteration to every hidden state produced by the network during the forward pass. Overall, the proposed BPTT learning algorithm differs from traditional BPTT-like implementations in two ways.

- Firstly, it includes a user-specified special parameter for controlling nonlinearity in the gradient calculation. Such a parameter is pivotal since it is closely related to the model's convergence properties.
- Secondly, the proposed BPTT method employs a mechanism for aggregating the gradients necessary to update the model's weights governing the recurrent reasoning block. This approach aims to mitigate the impact of vanishing gradient issues during the estimation of the inner weight matrix by consolidating gradients that have not yet vanished. Consequently, if the norm of the gradient vector at a particular iteration falls below a predetermined threshold, any changes induced by that gradient will be disregarded when aggregating the gradient vectors.

The remainder of this paper is organized as follows. Section 2 reviews relevant state-of-the-art learning algorithms for the FCM family. Section 3 outlines the r-LTCN model, including a detailed toy example illustrating its reasoning process. Section 4 brings forward the main contribution of our paper — the revised BPTT learning method. Section 5 is devoted to the empirical study, in which we test and compare the new approach with state-of-the-art learning methods for the studied class of models. Section 6 concludes the paper.

2. Literature review

FCMs and the family of models derived from them have become indispensable in many practical domains. This class of models is characterized by high predictive power and convenient properties like interpretability and ease of parametrization. These sought-after features contributed to their successful application to many tasks, including classification, forecasting, and recommender systems [9]. As further model refinements were proposed, delivering adequate learning algorithms capable of detecting data regularities and turning them into proper model parameters became imperative.

To properly present and position our contribution delivered in this paper, we shall discuss the existing solutions to FCM (and related models) training. In that regard, the literature reports on three groups of approaches: heuristic-based, inverse-based, and gradient-based methods.

Heuristic-based approaches. The growth of the popularity of FCM-related models overlapped with a massive increase in the popularity of heuristic optimization. It was only natural to test how well heuristic optimizers like Genetic Algorithm (GA), Particle Swarm Optimization (PSO) [10], and others perform in the task of weight matrix learning. In turn, the literature devoted to FCMs offers a wide range of solutions utilizing heuristic-based approaches.

Heuristic-based approaches combined with Fuzzy Cognitive Maps (FCMs) have been prevalent in the literature for over 20 years [11]. Based on the review of FCM extensions conducted by Schuerkamp and Giabbanelli [12], we can confidently assert that heuristic optimization continues to be, and will remain, a widely used method for learning FCM-related models. Therefore, in the experimental section of this paper, we will employ Genetic Algorithms (GA) as one of the state-of-the-art models for comparison with our new method.

The popularity of heuristic-based methods should not cloud our judgment about their properties. First and foremost, they cannot guarantee to find the optimal solution. The idea behind them is to offer a solution "good enough", which is enough for many practical problems. The second issue with heuristic optimization is their uncertainty in efficiency. If we are unlucky, the time needed to find this "good enough" solution can be large [13]. From the point of view of the FCM family of models, the third essential flaw needs to be mentioned: the best candidate solution from one iteration to another can differ a lot. This is because, in most algorithms, the best candidate solution can be sourced from many individuals in a population.

Inverse-based approaches. This approach to FCM learning was first introduced in the paper by Vanhoenshoven et al. [14]. It was delivered to tackle the critical flaws of heuristic-based systems listed in the previous paragraph. The authors proposed a deterministic learning algorithm that uses the Moore–Penrose [15] inverse to compute the weight matrix. In the cited work, maps were applied to system simulation. The essential advantage of inverse-based learning is that it does not require laborious adjustment of parameters, which is especially characteristic of heuristic-based approaches. The second fundamental property of inverse-based learning is that it is deterministic and extremely fast compared to heuristic-based methods. All these advantageous features are supplemented with a satisfying accuracy of the model's forecasts.

Other works have brought forward FCM-rooted models such as Short-term Cognitive Networks (STCN) [4] and the Long Short-term

Table 1
Properties of state-of-the-art methods for training FCM-based architectures using gradient descent techniques.

Authors	Training method	Task	Model type	Optimization goal
Chen et al. [18]	Hybrid: gradient descent and heuristics	Simulation	FCM	Simulation error
Wu et al. [19]	Hybrid: evolutionary optimization and regularized leader proximal algorithm	Gene Regulatory Network reconstruction	OFCM	Map structure and prediction error
Sabahi and Stanfield [20]	Error back-propagation	Simulation	FCM	Simulation error
Karatzinis and Boutalis [21]	Hybrid: gradient descent and least squares method	Classification and time series prediction	FCN	Prediction error
Karatzinis et al. [22]	Hybrid: gradient descent and least squares method	Classification	FCN	Prediction error
Qin et al. [8]	Error back-propagation	Time series prediction	DAFCM	Prediction error
Wang et al. [23]	Gradient descent	Time series prediction	DFCM	Prediction error

Cognitive Networks (LSTCN) [16]. All these models utilized a regression-based learning scheme combined with other forms of learning. The authors showed that it can succeed in various predictive tasks, including classification and forecasting. Similarly, an important classification-oriented model was delivered by Wu et al. [17]. They proposed a new FCM extension to the Broad FCM model, which was supplemented with a pseudoinverse-based learning method.

Gradient-based approaches. The third group of methods for learning FCM-based models utilizes gradient-based learning. The method proposed in this paper belongs to this family. Therefore, we will address it in greater detail.

Let us first mention a few recently developed methods that are on the borderline of this group and the group of heuristic optimizers. Chen et al. [18] proposed an adaptation of the FCM model to mimic the actions of the situation awareness global assessment technique. Their proposal relied on translating this domain-specific technique into the world of computations. In that regard, several modifications had to be introduced, combined with a dedicated meta-heuristic learning method. This heuristic search was subsequently strengthened with gradient-based optimization. In particular, the global exploration part was done using PSO update formulas, but the local exploitation part was implemented using the gradient descent method. An analogous research methodology can be found in the study by Wu et al. [19], where the authors addressed the need to adapt an FCM model to the online learning scenario. They proposed a hybrid model named OFCM, which fuses evolutionary optimization with an online gradient descent method known from online learning. The proposed system efficiently tackles large-scale online processing tasks.

On the other hand, an approach to FCM learning patterned on the mechanisms of error backpropagation was delivered by Sabahi and Stanfield [20]. Their method was termed FCM learning Algorithm and Visualization and contains three phases. The first uses simple rules to obtain the system bias, while the second phase aims to acquire the FCM weight matrix. Finally, the third phase produces individual concept biases. Their approach, in its essence, is similar to the neural network error backpropagation mechanism. This method was applied to data modeling, and its effective performance across various predictive tasks has yet to be tested.

Karatzinis and Boutalis [21] were working on pattern classification tasks using an FCM extension named Fuzzy Cognitive Network (FCN). Due to the nature of their study, they wanted to utilize classification errors to navigate the map training procedure. This has led them to develop a method combining a gradient-descent-like procedure that uses either a linear or a bilinear parametric model of the network and a least squares method for the estimation of the functional weights. The model being constructed (the FCN) has functional weights, which is not a typical assumption. Furthermore, the method was delivered specifically for pattern classification. The same research team has recently published a paper [22] in which FCNs were also applied to classification tasks. This time, the authors tested optimizers implemented for standard Convolutional Neural Network Models.

Gradient-based techniques are inherently utilized to minimize a task-specific cost function. As a result, this learning paradigm derives a weight matrix in an error-driven manner. Naturally, the specifics of the task (classification or regression) influence the model construction. Since we address general regression tasks in this paper, let us now mention some learning approaches specifically developed for this task.

As Orang et al. [24] report in their survey, Gregor and Groumpos [25] filled in the gaps in the literature by providing introductory insights into how an FCM model trained using the backpropagation through time algorithm (BPTT) would perform in data simulation tasks. BPTT [26] is a variant devoted to recurrent models specific to the tasks of sequential data processing. The primary rationale of BPTT is that recurrent neural models can be unfolded into a feed-forward neural network. The errors are computed and utilized to update the weights using a standard backpropagation technique. One crucial difference is that equivalent weights must be identical in all instances (i.e., the network is copied into a new instance at each time step). Therefore, we require the aforementioned correspondence. It comes without surprise that this model is specific to sequential data, so each time step of the unfolded network can be represented as a layer. The network catches then a window of time-dependent observations, and the computation of a step $t+1$ depends on the values in the step t [27]. Unfortunately, the literature does not offer any more insights into the applications of BPTT to optimize FCM-based models.

Finally, let us mention that other extensions to the FCM architecture using the error backpropagation method have been proposed. A very recent FCM-based architecture suitable for time series forecasting tasks was delivered by Qin et al. [8]. It was termed deep attention FCM (DAFCM) and relied on inserting blocks mimicking FCM's behavior into an LSTM model. Similarly, Wang et al. [23] proposed an FCM extension called deep FCM (DFCM) for multivariate time series forecasting. Their model combines a fully connected FCM to model relationships among the concepts and a recurrent neural network to represent exogenous factors influencing the system dynamics.

Table 1 summarizes essential properties of the gradient-based optimization methods for FCM-based models present in the literature. In the table, the listed approaches may be accounted to the same family, i.e., gradient-based methods tackling supervised learning tasks. The majority of the proposed methods combine a gradient-based optimizer with another optimization algorithm. The need for combination usually comes from the hybridization with another architecture, such as neural networks. None of the described methods directly use BPTT to train the FCM-based architectures.

3. Recurrence-aware long-term cognitive networks

This section describes the r-LTCN model [28], which further strengthens the predictive capability of LTCNs used for pattern classification and multi-output regression. In this model, the weights are

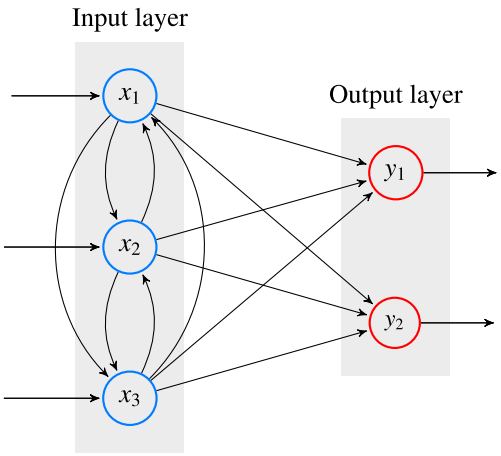


Fig. 1. Recurrence-aware LTCN architecture for a prediction problem with three inputs (x_1, x_2, x_3) and two outputs (y_1, y_2).

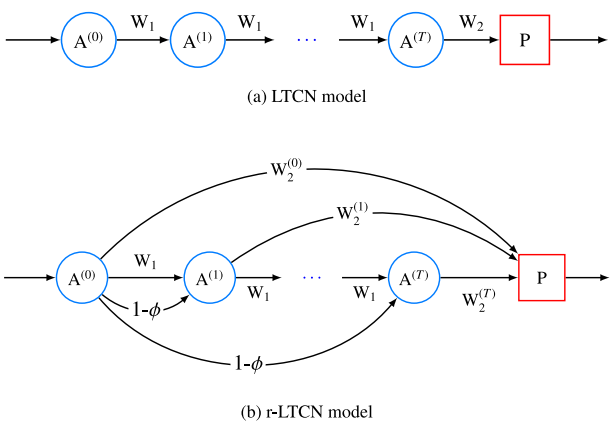


Fig. 2. Decision-making process of LTCN-based models with unrestricted weights: (a) LTCN model that uses a reasoning rule without recurrence-aware connections and (b) r-LTCN model that uses a quasi-nonlinear reasoning rule with recurrence-aware connections.

not confined to the $[-1, 1]$ interval, following the same intuition of the LTCN formalism. However, what makes the r-LTCN model powerful is that it uses all temporal states generated by the recurrent neural network during reasoning, not only the last network's state, as happens with both LTCN and FCM models. As a result, decision-making will always be possible, even if the network converges to a unique fixed-point attractor [3], where the network will produce the same output regardless of the input used to activate the neurons. Another distinctive aspect of r-LTCNs compared with traditional FCM models is that they use a parameterized reasoning mechanism that allows for controlling the system's nonlinearity.

Fig. 1 portrays an example of the architecture of this recurrence-aware LTCN-based model for a prediction problem involving five variables, where three are deemed inputs while the others are regarded as outputs.

In order to mathematically describe the r-LTCN model, let us assume that we have a prediction problem (either classification or multi-output regression) having n input features and m outputs. For simplicity, we can assume that all variables are numerical. When operating in a mini-batch mode, the network takes an initial activation matrix $\mathbf{A}^{(0)}$ composed of k instances and n input variables. It produces the prediction matrix $\mathbf{P}$ with the predictions for m composed by k instances and m output variables. Therefore, these matrices have dimensions $k \times n$ and $k \times m$, respectively. During reasoning, the network produces T activation vectors (or states) $\mathbf{A}^{(t)}$ that symbolize the network's hidden states as depicted below:

$$\mathbf{A}^{(t)} = \phi \cdot f\left(\mathbf{A}^{(t-1)} \cdot \mathbf{W}_1\right) + \left(1 - \phi\right) \cdot \mathbf{A}^{(0)} \tag{1}$$

where $\phi \in [0, 1]$ stands for the nonlinearity coefficient [5] defined by the user to control the balance between the neurons' initial state $\mathbf{A}^{(0)}$ and the previous hidden state $\mathbf{A}^{(t-1)}$, whereas $\mathbf{W}_1$ is a learnable $n \times n$ matrix defining the interaction between neurons. It should be mentioned that the ϕ parameter shall be selected in a standard cross-validation procedure, in which one repeats model training for different values and selects one that yields the highest results. The sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ provides the network with nonlinearity, thus allowing the r-LTCN model to capture complex patterns.

It is relevant to stress that Eq. (1) is used to update the neuron's activation values in the input layer (also referred to as recurrent block). After the recurrent block performs a predefined number of iterations T, the prediction matrix $\mathbf{P}$ is computed in the decision-making layer as follows:

$$\mathbf{P} = \mathbf{H}^{(T)} \cdot \mathbf{W}_2. \tag{2}$$

such that $\mathbf{W}_2$ is an $n(T+1) \times m$ is a matrix that connects the hidden states $\mathbf{A}^{(t)}$ with the output neurons, which use a linear activator. Furthermore, $\mathbf{H}^{(T)}$ is a $k \times n(T+1)$ matrix resulting from the recursive horizontal concatenation of the initial condition $\mathbf{A}^{(0)}$ encoding the input and the T temporal states produced during reasoning as depicted below:

$$\mathbf{H}^{(T)} = \left(\mathbf{A}^{(0)} | \mathbf{A}^{(1)} | \mathbf{A}^{(2)} | \ldots | \mathbf{A}^{(T-2)} | \mathbf{A}^{(T-1)} | \mathbf{A}^{(T)}\right). \tag{3}$$

Fig. 2 depicts the decision-making process of the LTCN model, which implements neither the recurrence-aware mechanism nor quasi-nonlinear reasoning, and the r-LTCN model described above. Such a representation makes it evident that r-LTCNs are much more powerful and expressive than the LTCNs and the traditional FCM-based models.

Let us briefly review the LTCN inference mechanism for a hypothetical case study with the following input:

$$\mathbf{W}_1 = \begin{bmatrix} 0.2 & 0.1 & 0.8 \\ 0.5 & 0.7 & 0.2 \\ 0.1 & 0.3 & 0.3 \end{bmatrix}, \mathbf{W}_2^{(0)} = \begin{bmatrix} 0.4 & 0.8 \\ 0.5 & 0.1 \\ 0.2 & 0.2 \end{bmatrix}, \mathbf{W}_2^{(1)} = \begin{bmatrix} 0.1 & 0.5 \\ 0.2 & 0.4 \\ 0.9 & 0.7 \end{bmatrix}$$

$$\mathbf{W}_2^{(2)} = \begin{bmatrix} 0.5 & 0.1 \\ 0.9 & 0.7 \\ 0.4 & 0.8 \end{bmatrix}, \mathbf{W}_2^{(3)} = \begin{bmatrix} 0.2 & 0.4 \\ 0.1 & 0.6 \\ 0.1 & 0.4 \end{bmatrix}, \mathbf{W}_2^{(4)} = \begin{bmatrix} 0.4 & 0.6 \\ 0.5 & 0.2 \\ 0.3 & 0.9 \end{bmatrix}$$

$$\mathbf{W}_2^{(5)} = \begin{bmatrix} 0.6 & 0.8 \\ 0.6 & 0.0 \\ 0.1 & 0.2 \end{bmatrix}, \mathbf{A}^{(0)} = [0.1 \;\; 0.1 \;\; 0.8], \;\; \phi = 0.6, \;\; T = 5.$$

We assume the weight matrices given above were obtained in the training procedure. Next, we proceed with the demonstration of the model's inner workings such that values are rounded to two decimal places. In the initial step, we ought to calculate intermediate activation values for $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(5)}$. In order to do so, we execute Eq. (1) as follows:

$$\mathbf{A}^{(1)} = \phi \cdot f(\mathbf{A}^{(0)} \cdot \mathbf{W}_1) + (1 - \phi) \cdot \mathbf{A}^{(0)} = [0.36 \;\; 0.39 \;\; 0.67].$$

For the consecutive activation vectors, we obtain:

$$\mathbf{A}^{(2)} = [0.49 \;\; 0.53 \;\; 0.65],$$

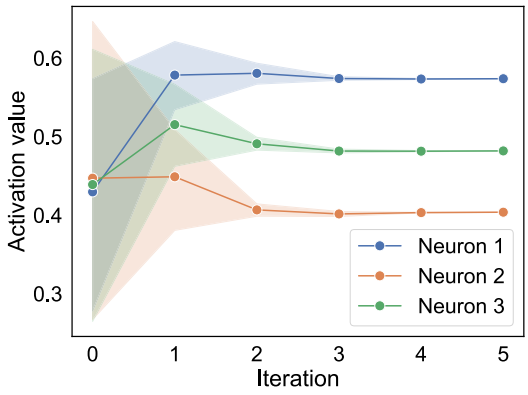
$$\mathbf{A}^{(3)} = [0.56 \;\; 0.60 \;\; 0.66],$$

$$\mathbf{A}^{(4)} = [0.59 \;\; 0.64 \;\; 0.67],$$

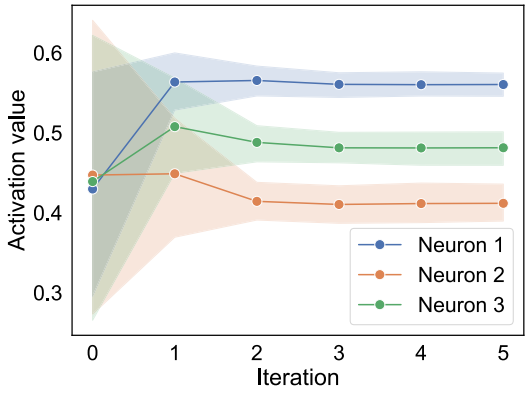
$$\mathbf{A}^{(5)} = [0.61 \;\; 0.66 \;\; 0.68].$$

To compute the prediction matrix $\mathbf{P}$ using the weights and state vectors, let us rewrite Eq. (2) as follows:

$$\mathbf{P} = \mathbf{H}^{(T)} \cdot \mathbf{W}_2 = \sum_{t=0}^{T} \mathbf{A}^{(t)} \cdot \mathbf{W}_2^{(t)} = [3.78 \;\; 4.57].$$



(a) $\phi = 1.0$



(b) $\phi = 0.9$

Fig. 3. Reasoning process of the quasi-nonlinear reasoning mechanism for a three-neuron model using randomly generated activation vectors.

However, r-LTCNs are not just more powerful than LTCNs and FCMs because they involve more learnable parameters. The truth is that the ϕ parameter regulating the quasi-nonlinear reasoning mechanism plays a pivotal role in avoiding the network's convergence to a unique fixed-point attractor. If this undesirable situation emerges, the recurrent network will produce the same outputs for every possible initial activation vector, rendering decision-making impossible in some cases. Fig. 3 shows the activation values for a three-neuron LTCN model using randomly generated initial activation vectors and quasi-nonlinear reasoning but excluding the recurrence-aware connections. While $\phi = 1.0$ leads to a unique fixed point, making the network's outputs independent from its inputs, $0 < \phi < 1$ produces input-dependent responses.

Although the r-LTCN model boasts enhanced representation capabilities, its main challenge lies in its training process. The authors in [28] proposed a two-step learning algorithm that used unsupervised learning to compute the $\mathbf{W}_1$ and supervised learning rule based on the Moore–Penrose inverse to calculate $\mathbf{W}_2$. The limitation of this approach is that both learning processes are detached in the sense that they are performed sequentially. To our knowledge, no other attempts have been reported to train r-LTCNs in a fully supervised manner. This might be motivated by the following challenges. On the one hand, training a recurrent neuronal network that performs like a closed system using an algorithm like BPTT is more difficult than training a recurrent neural system that receives a new input in each iteration. This happens because the gradient often vanishes within the recurrent neural block, conferring the learning burden to the recurrence-aware connections. On the other hand, we need to consider the particularities of the quasi-nonlinear reason rule that involves a hyperparameter that affects the partial derivatives associated with the neurons' outputs.

4. Proposed BPTT algorithm

This section presents the primary theoretical contribution of our paper: a BPTT-like algorithm designed to train the r-LTCN model discussed in the preceding section. The proposed BPTT algorithm differs from traditional approaches in three key aspects. Firstly, it operates under the assumption that the r-LTCN model does not receive fresh input in each iteration. As such, the system's current state relies solely on the state produced by the network in the previous iteration. Secondly, it adjusts the partial derivative of the neurons' outputs concerning their raw activation values to use the regularized signal responsible for the quasi-nonlinear reasoning rule when updating the weights in the recurrent block. Thirdly, it incorporates a simple yet effective procedure to mitigate the adverse effects of vanishing gradient issues on the learning process.

4.1. Forward pass

Let $\mathbf{X}$ and $\mathbf{Y}$ be the input and output data with dimensions $k \times n$ and $k \times m$, respectively, where n is the number of input variables, m denotes the number of output variables and k represents the number of training instances. Moreover, we will assume that the sigmoid function is the activation function used to clip concepts' activation values.

The forward pass updates the activation matrix $\mathbf{A}^{(t)}$ at each iteration t using Eq. (1) and computes the predictions $\mathbf{P}$ using Eq. (2). The initial activation matrix is defined as $\mathbf{A}^{(0)} = \mathbf{X}$. Notice that this model does not involve any bias terms as they would affect the system's interpretability.

4.2. Backward pass

In the backward pass, we must first calculate the loss function to measure the model's prediction error. For simplicity, we will adopt the mean square error as shown below:

$$\mathbf{L} = (\mathbf{P} - \mathbf{Y})^2. \tag{4}$$

For this measure, the partial derivative of the loss function w.r.t. the neurons' outputs in the decision-making (output) layer is depicted in the following equation:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{P}} = -2 \cdot (\mathbf{Y} - \mathbf{P}). \tag{5}$$

Therefore, the gradient to update the $\mathbf{W}_2$ matrix connecting the system's temporal states with the output layer can be straightforwardly computed as follows:

$$\nabla_2 = \left(\mathbf{H}^{(T)}\right)^\top \cdot \frac{\partial \mathbf{L}}{\partial \mathbf{P}} \tag{6}$$

Aiming to compute the $\mathbf{W}_1$ matrix in the recurrent block, we will rely on the following chain rule:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathbf{L}}{\partial \mathbf{A}^{(t)}} \times \frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}} \times \frac{\partial \bar{\mathbf{A}}^{(t)}}{\partial \mathbf{W}_1}. \tag{7}$$

When computing the gradients used to update $\mathbf{W}_1$, we must distinguish between two cases for transparency. The first one happens when $t = T$, which means that we need to bring the error signal coming from the output layer to the recurrent block. The second case assumes that the error signal is already included in the gradient calculations.

Case 1. When $t = T$, we have the following:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{A}^{(t)}} = \frac{\partial \mathbf{L}}{\partial \mathbf{P}} \cdot \left(\mathbf{W}_2^{(T)}\right)^\top \tag{8}$$

such that $\mathbf{W}_2^{(T)} \subseteq \mathbf{W}_2$ that connects the recurrent block's last iteration with the decision-making block.

Case 2. When $t < T$, we have the following:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{A}^{(t)}} = \frac{\partial \mathbf{L}}{\partial \mathbf{A}^{(t-1)}} \cdot \mathbf{W}_1. \tag{9}$$

Subsequently, we need to compute $\frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}}$ in Eq. (8) acknowledging that the system's states are calculated as a weighted linear combination of the neurons' activation states in the previous iteration and their initial activation values. Such a weighted combination is controlled by the nonlinearity coefficient ϕ in Eq. (1). Therefore, $\frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}}$ is given by:

$$\frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}} = \phi \cdot f\left(\mathbf{A}^{(t-1)} \cdot \mathbf{W}_1\right) \cdot \left(1 - f\left(\mathbf{A}^{(t-1)} \cdot \mathbf{W}_1\right)\right). \tag{10}$$

such that $\bar{\mathbf{A}}^{(t)} = \mathbf{A}^{(t-1)} \cdot \mathbf{W}_1$. It holds that

$$\frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}} = \frac{\partial f\left(\bar{\mathbf{A}}^{(t)}\right)}{\partial \bar{\mathbf{A}}^{(t)}} = \frac{\partial}{\partial \bar{\mathbf{A}}^{(t)}}\left[\phi \cdot \frac{1}{1+e^{-\bar{\mathbf{A}}^{(t)}}}\right] = \phi \cdot \frac{e^{-\bar{\mathbf{A}}^{(t)}}}{(1+e^{-\bar{\mathbf{A}}^{(t)}})^2}$$

and we also know that

$$\frac{e^{-\bar{\mathbf{A}}^{(t)}}}{(1+e^{-\bar{\mathbf{A}}^{(t)}})^2} = \frac{1}{1+e^{-\bar{\mathbf{A}}^{(t)}}} \cdot \frac{e^{-\bar{\mathbf{A}}^{(t)}}}{1+e^{-\bar{\mathbf{A}}^{(t)}}} = \frac{1}{1+e^{-\bar{\mathbf{A}}^{(t)}}} \cdot \left(1 - \frac{1}{1+e^{-\bar{\mathbf{A}}^{(t)}}}\right).$$

Therefore, we have the following:

$$\frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}} = \phi \cdot f\left(\bar{\mathbf{A}}^{(t)}\right) \cdot \left(1 - f\left(\mathbf{A}^{(t-1)} \cdot \mathbf{W}_1\right)\right). \tag{11}$$

Finally, $\frac{\partial \bar{\mathbf{A}}^{(t)}}{\partial \mathbf{W}_1} = \mathbf{A}^{(t-1)}$ since the neuron's raw activation values only depend on the $\mathbf{W}_1$ weight matrix and the neurons' activation values in the previous iteration.

At this point, all components in Eq. (8) have already been computed, which means that we can update the $\mathbf{W}_1$ matrix. However, it must be noticed that contrary to what happens in feed-forward neural networks where weights change from one iteration to another, the $\mathbf{W}_1$ matrix must remain unchanged in all iterations. Using the gradient signal that arrives at the first abstract layer would be a sub-optimal choice for three reasons. Firstly, such a gradient signal would describe the changes to be made to the weights connecting the first abstract layer (containing the neurons' initial activation values) with the second one (containing the neurons' activation values in the first iteration). Secondly, it assumes that the weights connecting the deeper abstract layers would be modified using different gradient signals. Thirdly, the gradient signals arriving at the first abstract layers might have vanished or exploded.

To address both issues, we propose a highly effective procedure that involves two steps. Firstly, we clip all gradient signals to confine them to the $[-U, U]$ intervals, such that $U > 0$ is a parameter. In all simulations conducted in the following section, U was set to 100. However, we did notice that the clipping method was never called. Secondly, we aggregate all gradient signals having a norm larger than zero in order to obtain a "consensus gradient" to be used to update the $\mathbf{W}_1$ matrix in a single step. This approach was inspired by the BPTT method delivered by Grau et al. [29] that employed a similar approach to deal with variable-length sequences. Fig. 4 illustrates the rationale of our method for an r-LTCN model performing 10 iterations, which can be unfolded into the same number of abstract layers. In this example, only the gradients associated with the abstract layers falling within the gray region will be aggregated to obtain the consensus gradient signal.

The computation of the consensus gradient used to update the $\mathbf{W}_1$ matrix is formalized below:

$$\nabla_1 = \frac{1}{T_\beta - 1} \sum_t \left(\frac{\partial \mathbf{L}}{\partial \mathbf{A}^{(t)}} \times \frac{\partial \mathbf{A}^{(t)}}{\partial \bar{\mathbf{A}}^{(t)}} \times \frac{\partial \bar{\mathbf{A}}^{(t)}}{\partial \mathbf{W}_1}\right), \tag{12}$$

such that $0 < T_\beta \leq T$ gives the number of gradient signals with a non-zero norm (after clipping) that flows within the recurrent block. In other words, the zero-norm gradients will be ignored when computing the consensus gradient.

In our paper, we resorted to the Adam optimizer to update the learnable matrices as shown below:

$$\mathbf{W}_i = \mathbf{W}_i - \alpha \cdot \frac{m(\mathbf{W}_i)}{\sqrt{v(\mathbf{W}_i)} + \epsilon} \tag{13}$$

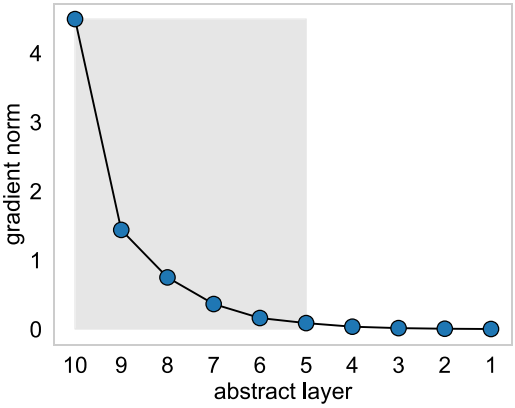


Fig. 4. Aggregation of gradient signals having a non-zero norm in order to obtain a consensus gradient for an r-LTCN model.

Table 2
Description of datasets used in the experiments.

ID	Dataset	Features	Instances
D1	collins	21	500
D2	ecoli	7	336
D3	flags	28	194
D4	glass	9	214
D5	led7digit	7	500
D6	libras	26	360
D7	mfeat-fourier	76	2000
D8	mfeat-zernike	25	2000
D9	parkinsos	22	195
D10	pima	8	768
D11	planning-relax	12	182
D12	plant-margin	64	1600
D13	plant-shape	64	1600
D14	plant-texture	64	1599
D15	segment	19	2310
D16	wdbc	30	569
D17	wine	13	178
D18	winequality-red	11	1599
D19	winequality-white	11	4898
D20	yeast	8	1484

such that

$$m(\mathbf{W}_i) = \frac{\beta_1 \cdot m(\mathbf{W}_i) + (1-\beta_1) \cdot \nabla_i}{1 - (\beta_1)^j} \tag{14}$$

$$v(\mathbf{W}_i) = \frac{\beta_2 \cdot v(\mathbf{W}_i) + (1-\beta_2) \cdot \nabla_i^2}{1 - (\beta_2)^j} \tag{15}$$

where ∇_i denotes the gradient for the ith weight matrix, $m(\mathbf{W}_i)$ and $v(\mathbf{W}_i)$ are the first and second moment estimates, respectively. Moreover, β_1 and β_2 are the hyperparameters of the Adam optimizer, ϵ is a small value to prevent division by zero, and j is the current batch number.

5. Experiments

To study the effectiveness of the proposed BPTT learning algorithm, we will resort to 20 real-world datasets taken from the Keel repository [30]. These datasets are described by numerical features and were originally devoted to pattern classification, which is not the targeted learning task. To transform these datasets into multi-output prediction problems, we removed the decision classes and arbitrarily used the last two features as target variables and the remaining ones as inputs. As such, we have multivariate inputs and multivariate (bivariate) outputs that fit the desired learning task. Table 2 outlines the main features of datasets processed in our empirical study.

In the first experiment, we will study the performance of the LTCN-based architectures trained with different learning algorithms. These architectures are depicted in Fig. 2 and differ on whether they implement the recurrence-aware mechanism. In that way, we can explore the effect of using different learning methods while performing an ablation study on the network's components. However, it is worth mentioning that both LTCN-based architectures employ the quasi-nonlinear reasoning rule in Eq. (1) to escape from the unique fixed point. As for the learning algorithms, we consider Genetic Algorithms (GA), Moore–Penrose Inverse (MP), and the proposed BPTT algorithm. The GA is representative of the metaheuristic-based learning algorithms, while the MP method is representative of the regression-based learning algorithms.

To facilitate the discussion of results, let us introduce the naming convention `r-LTCN-X` to encode the LTCN-based architectures and the explored learning algorithms (`MP`, `GA`, and `BPTT`). The presence of the first term "`r`" indicates that the model fine-tunes the recurrence-aware connections. In contrast, `LTCN-X` means that the recurrence-aware connections are ignored. The second component indicates that we employ the `LTCN` model, while the third component informs that the model was trained with the `X` learning algorithm.

For each model, we perform nested 5-fold cross-validation to fine-tune the pertinent hyperparameters. For the proposed BPTT method, we fine-tune the batch size (1, 8, 16, and 32) and the learning rate (0.001, 0.005, and 0.01). The GA method uses 100 chromosomes and 50 generations, while the crossover and mutation probabilities are set to 0.9 and 0.1, respectively. The mentioned parameter setup is often used in the literature, c.f. [31,32]. More aggressive settings are possible at the expense of significantly increasing the training time. The MP learning method is parameterless. In all cases, we fine-tune the nonlinearity degree (0.2, 0.5, and 0.8) attached to the quasi-nonlinear reasoning rule. We also train a linear regression model (without an intercept) as a baseline. The decision not to fit the intercept is motivated by the fact that all components in FCM-rooted models must have a clear interpretation. As for the performance metrics, we use the Root Mean Squared Error (RMSE), the Mean Absolute Percentage Error (MAPE), and the Symmetric Mean Absolute Percentage Error (SMAPE).

Fig. 5 portrays the average RMSE scores reported by each model on the test set after performing nested 5-fold cross-validation. This figure allows us to compare the models trained with different learning algorithms and the baseline. The results show that `r-LTCN-BPTT` and `LTCN-BPTT` are the best-performing models, while the ones trained with the GA learning algorithm reported the worst results overall. In the most drastic case, it turned out to be three times worse than the BPTT approach on average. What is more, the dispersion of the achieved results is the highest for the GA-trained models. Therefore, on top of high errors, their performance is the least stable. The MP learning algorithm is the second-best learning strategy, occasionally outperformed by the baseline.

Table 3 displays the p-value reported by the Wilcoxon signed-rank test for each pairwise comparison where the proposed `r-LTCN-BPTT` model is the control method. Moreover, we report the negative (R^-) and the positive (R^+) ranks, the corrected p-values as computed by Holm's post-hoc method, and whether the null hypothesis is rejected or not for a 95% confidence interval. Note that the Holm–Bonferroni correction method is used to control the family-wise error rate resulting from multiple comparisons.

The p-values from Holm's post-hoc method suggest rejecting the null hypotheses in all cases. It confirms that the differences between `r-LTCN-BPTT` and the remaining variants are statistically significant. In addition, such differences translate into the control method producing smaller prediction errors since this algorithm reports fewer negative ranks overall.

Additionally, the average MAPE and SMAPE scores are reported in Figs. 6 and 7, respectively. The results reaffirm that `r-LTCN-BPTT` and `LTCN-BPTT` outperform the other methods in our empirical study.

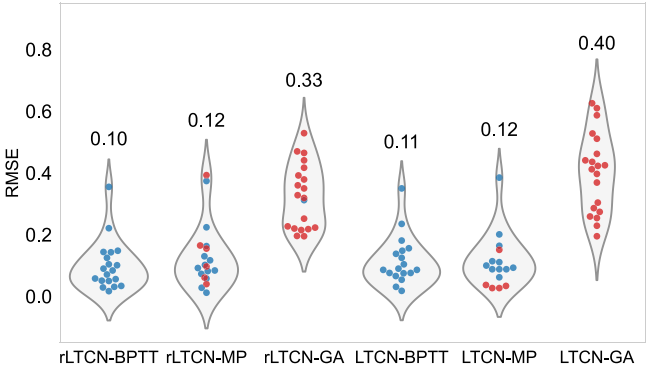


Fig. 5. RMSE scores obtained for each dataset using different models (LTCN architecture trained with a learning algorithm). Red dots indicate that the performance worsened compared to the baseline, while blue dots indicate the performance improved or remained unchanged. The average score of each model is reported on the top of each violin plot.

Table 3
Wilcoxon signed-rank pairwise test concerning the RMSE metric. The Holm–Bonferroni correction is used as the post-hoc method, while `r-LTCN-BPTT` learning algorithm is the control method.

Algorithm	p-value	R^-	R^+	Holm	H_0
Baseline	1.31E−04	19	0	3.94E−04	Reject
rLTCN-MP	1.47E−03	15	3	2.18E−03	Reject
rLTCN-GA	1.91E−06	20	0	1.14E−05	Reject
LTCN-BPTT	1.09E−03	16	1	2.18E−03	Reject
LTCN-MP	1.91E−06	20	0	1.14E−05	Reject
LTCN-GA	1.91E−06	20	0	1.14E−05	Reject

Table 4
Wilcoxon signed-rank pairwise test concerning the MAPE metric. The Holm–Bonferroni correction is used as the post-hoc method, while `r-LTCN-BPTT` learning algorithm is the control method.

Algorithm	p-value	R^-	R^+	Holm	H_0
Baseline	1.96E−04	18	0	5.87E−04	Reject
rLTCN-MP	8.93E−02	11	7	8.93E−02	Fail to reject
rLTCN-GA	1.91E−06	20	0	1.14E−05	Reject
LTCN-BPTT	1.18E−03	17	2	2.36E−03	Reject
LTCN-MP	1.91E−06	20	0	1.14E−05	Reject
LTCN-GA	1.91E−06	20	0	1.14E−05	Reject

Table 5
Wilcoxon signed-rank pairwise test concerning the SMAPE metric. The Holm–Bonferroni correction is used as the post-hoc method, while `r-LTCN-BPTT` learning algorithm is the control method.

Algorithm	p-value	R^-	R^+	Holm	H_0
Baseline	1.96E−04	18	0	5.88E−04	Reject
rLTCN-MP	6.41E−02	12	6	6.41E−02	Fail to reject
rLTCN-GA	1.91E−06	20	0	1.14E−05	Reject
LTCN-BPTT	2.08E−03	15	2	4.15E−03	Reject
LTCN-MP	1.91E−06	20	0	1.14E−05	Reject
LTCN-GA	1.91E−06	20	0	1.14E−05	Reject

The MP learning variants show competitive performance, consistently placing second. Tables 4 and 5 display the outcomes of the Wilcoxon signed-rank test using MAPE and SMAPE as error measures. In this comparison, the proposed `r-LTCN-BPTT` model serves as the control method for each pair. Based on the corrected p-values, the null hypotheses can be rejected for all cases, except for the comparison with `r-LTCN-MP`. Nonetheless, `r-LTCN-BPTT` wins the pairwise comparison ($R^- = 11$ and $R^- = 12$, respectively). Therefore, `r-LTCN-BPTT` demonstrates statistically significant superior performance in terms of MAPE and SMAPE scores.

Fig. 8 shows the hyperparameter sensitivity analysis for the essential parameters using RMSE as the performance metric. We compare the

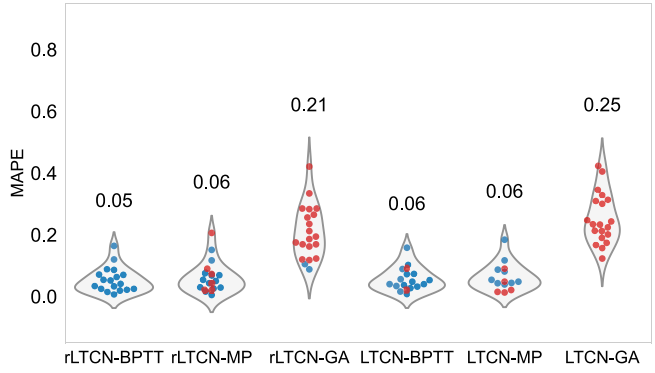


Fig. 6. MAPE scores obtained for each dataset using different models (LTCN architecture trained with a learning algorithm). Red dots indicate that the performance worsened compared to the baseline, while blue dots indicate the performance improved or remained unchanged. The average score of each model is reported on the top of each violin plot.

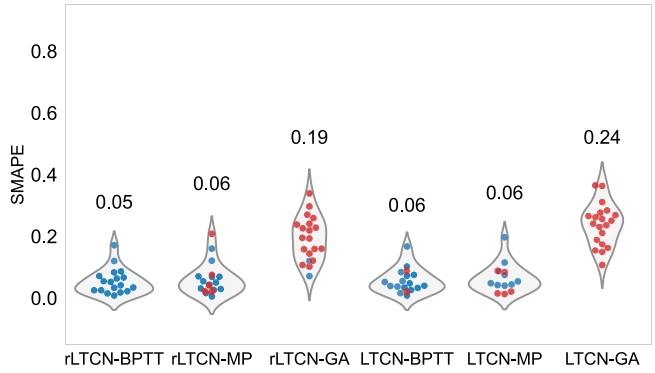
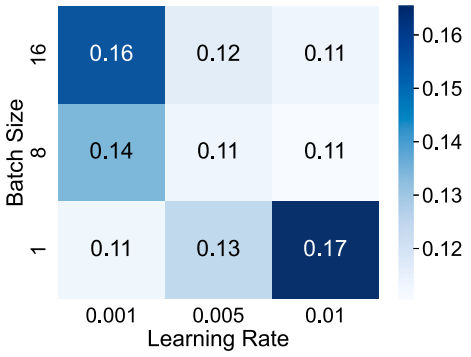
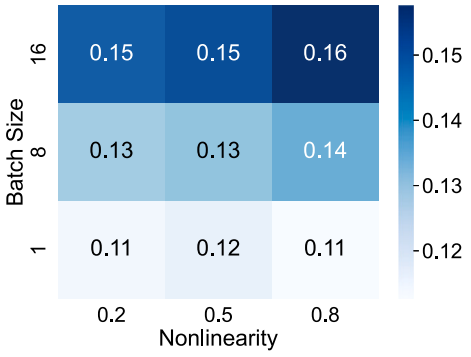


Fig. 7. SMAPE scores obtained for each dataset using different models (LTCN architecture trained with a learning algorithm). Red dots indicate that the performance worsened compared to the baseline, while blue dots indicate the performance improved or remained unchanged. The average score of each model is reported on the top of each violin plot.



(a) using ϕ=0.8



(b) using learning rate=0.001

Fig. 8. Average RMSE score on the validation data when varying the learning rate, batch size, and nonlinearity coefficient (ϕ).

results of an experiment where we changed the learning rate (values from the set {0.001, 0.005, and 0.01}), the batch size (1, 8, and 16), and the nonlinearity coefficient (0.2, 0.5, and 0.8). In the first plot, we set the nonlinearity coefficient to 0.8 and changed the batch size and learning rate. In the second, we set the learning rate to 0.001 and changed the batch size and the nonlinearity coefficient. The results show that several settings lead to accurate predictions for $\phi = 0.8$. However, smaller batch sizes are recommended when the learning rate is set to 0.001.

It should be stated that Fig. 8 shows the average RMSE for all 20 datasets studied in this paper. All in all, it must be emphasized that the average RMSE for different method parameter configurations is remarkably low. In the worst-case scenario, the average RMSE is 0.17. In the best-case scenario, the average RMSE is 0.11. The difference is just 0.06. Importantly, several combinations of method parameters provide low error values. Hence, we can conclude that the method's performance is robust to different settings.

Fig. 9 shows the distribution of weights computed by the BPTT algorithm for all datasets. The first observation is that the weights are centered at zero even without having a mechanism enforcing such behavior, which is a desired behavior that translates into a lower risk of overfitting and overflow situations. Overfitting implies that the model performs very well on the training data but poorly on the unseen data. Overflow is often an indicator of exploding gradients and might cause issues when evaluating the sigmoid function due to computing the exponential of very large weights. Zero-valued weights tend to regularize the processing flow, even if regularization is not explicitly implemented. Moreover, inspecting such a weight matrix is easier for a human being if one should be required to do so. Sparse matrices also allow us to pose conclusions regarding the impact of variables on processing.

Another interesting observation drawn from Fig. 9 is that weights in the $\mathbf{W}_1$ and $\mathbf{W}_2$ matrices behave similarly in most cases, which means that the model does not exclusively rely on the recurrence-aware connections to compute the predictions. Additionally, it can be noted that there are several problems where the $\mathbf{W}_2$ matrix tends to be more sparse than the $\mathbf{W}_1$ matrix (e.g., led7digit and mfeat-zernike).

6. Conclusions

In this study, we introduced a modified BPTT algorithm for training r-LTCN architectures in a fully supervised manner. Our motivation stemmed from the observation that although BPTT is a natural choice for training recurrent neural systems, its effectiveness had not been fully explored in the context of FCM-based models due to various challenges, including unique fixed-point attractors and the vanishing gradient problem.

Our proposed BPTT learning algorithm addresses these challenges by introducing some modifications to the original formalism. Firstly, it incorporates a user-specified parameter controlling nonlinearity in gradient calculation, a crucial factor closely linked to the convergence properties of the model. By allowing for control over the nonlinearity, we provide a means to enhance convergence and overall performance. Secondly, recognizing the closed system nature of r-LTCNs, wherein the model does not receive fresh input in each iteration, we devised a mechanism for aggregating gradients to update the $\mathbf{W}_1$ matrix. This approach mitigates the impact of vanishing gradients within the inner reasoning block, ensuring that the model continues to learn effectively despite these challenges. Thirdly, we highlighted the significance of

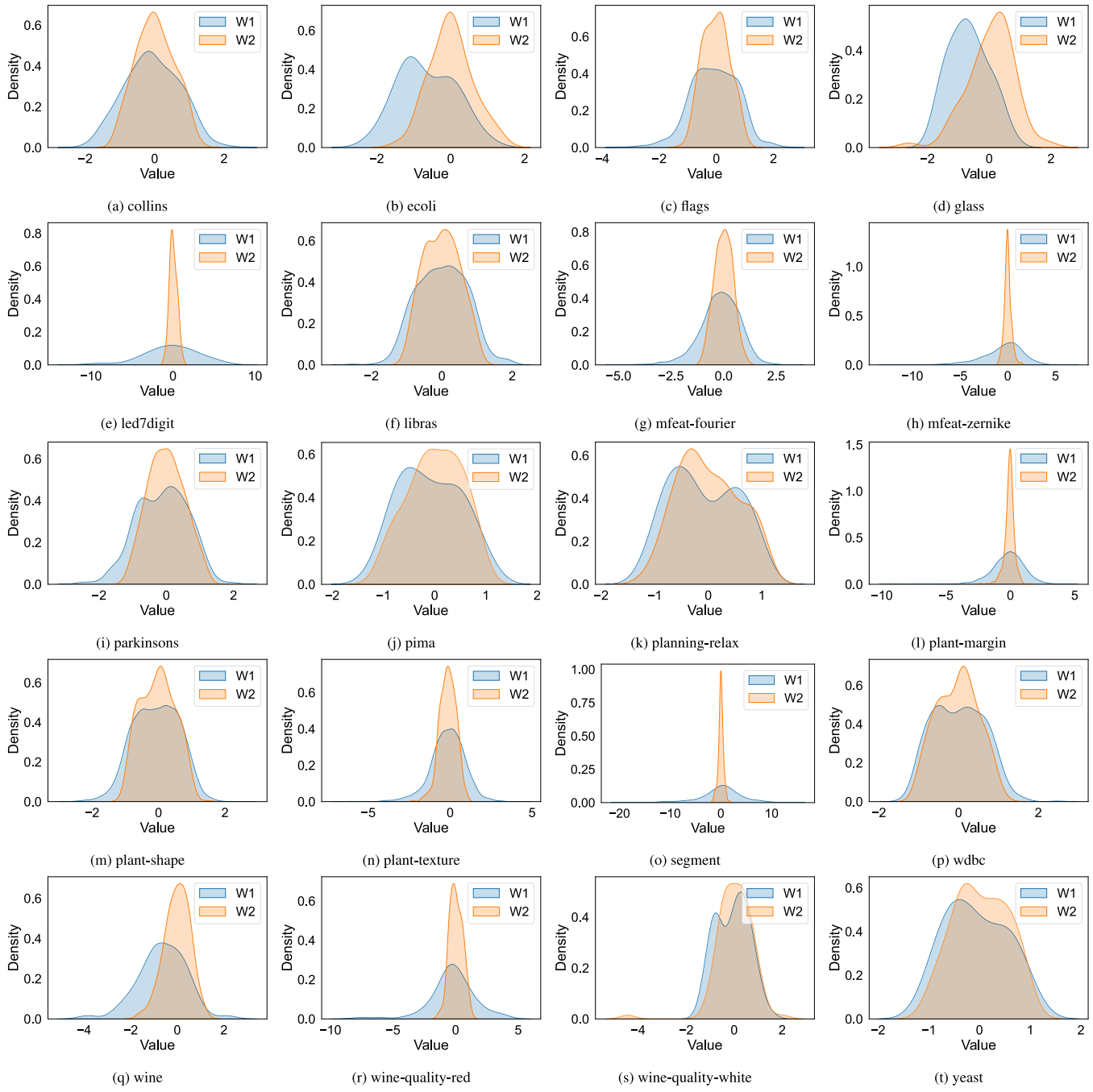


Fig. 9. Distribution of the weights learned by the proposed BPTT algorithm across various datasets. The centered distribution around zero suggests a reduced risk of overfitting and overflow situations, promoting regularization and easier interpretation. The similarities in behavior between $\mathbf{W}_1$ and $\mathbf{W}_2$ indicate the model's reliance on multiple matrices for predictions, with certain datasets exhibiting sparser $\mathbf{W}_2$ matrices compared to $\mathbf{W}_1$.

outer weights ($\mathbf{W}_2$ matrix) in the r-LTCNs' decision-making process. The recurrence-aware mechanism employed in r-LTCNs ensures that the error signal from the last iteration is propagated to every hidden state during the forward pass, thus emphasizing the importance of outer weights where gradients are less likely to vanish.

The empirical evaluation using 20 multi-output regression problems demonstrated the superiority of the proposed BPTT algorithm over other learning approaches. The experimental results statistically confirmed that r-LTCN-BPTT and LTCN-BPTT outperformed other variants, with GA-trained models exhibiting the highest errors and instability. Additionally, the analysis of hyperparameter sensitivity illustrated robust performance across various parameter configurations. The inspection of weight distributions further highlighted desirable properties, including centrality around zero and sparse matrices, which is indicative of regularization and interpretability. These findings underscore the effectiveness of the proposed BPTT algorithm in training LTCN-based models for multi-output regression tasks. In the future, we plan to study the performance in other prediction tasks, such as time series classification, which is a natural application area of the r-LTCN model.

CRediT authorship contribution statement

Gonzalo Nápoles: Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Agnieszka Jastrzebska:** Writing

– original draft, Validation, Methodology, Investigation, Formal analysis, Conceptualization. **Isel Grau:** Writing – original draft, Validation, Methodology, Investigation, Formal analysis, Conceptualization. **Yamisleydi Salgueiro:** Writing – original draft, Validation, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Y. Salgueiro would like to acknowledge the support provided by ANID Fondecyt, Chile Regular 1240293 and Basal National Center for Artificial Intelligence CENIA, Chile FB210017.

References

[1] Bart Kosko, Fuzzy cognitive maps, Int. J. Man-Mach. Stud. 24 (1) (1986) 65–75.
[2] Gonzalo Napoles, Frank Vanhoenshoven, Rafael Falcon, Koen Vanhoof, Nonsynaptic error backpropagation in long-term cognitive networks, IEEE Trans. Neural Netw. Learn. Syst. 31 (3) (2020) 865–875.
[3] István Á. Harmati, László T. Kóczy, On the convergence of input-output fuzzy cognitive maps, in: Rafael Bello, Duoqian Miao, Rafael Falcon, Michinori Nakata, Alejandro Rosete, Davide Ciucci (Eds.), Rough Sets, Springer International Publishing, Cham, 2020, pp. 449–461.
[4] Gonzalo Nápoles, Frank Vanhoenshoven, Koen Vanhoof, Short-term cognitive networks, flexible reasoning and nonsynaptic learning, Neural Netw. 115 (2019) 72–81.
[5] Gonzalo Nápoles, Isel Grau, Leonardo Concepción, Lisa Koutsoviti Koumeri, João Paulo Papa, Modeling implicit bias with fuzzy cognitive maps, Neurocomputing 481 (2022) 33–45.
[6] Marios Tyrovolas, X. San Liang, Chrysostomos Stylios, Information flow-based fuzzy cognitive maps with enhanced interpretability, Granul. Comput. (2023).
[7] Mabel Frias, Gonzalo Nápoles, Yaima Filiberto, Rafael Bello, Koen Vanhoof, Skipped nonsynaptic backpropagation for interval-valued long-term cognitive networks, in: Obdulia Pichardo Lagunas, Juan Martínez-Miranda, Bella Martínez Seis (Eds.), Advances in Computational Intelligence, Springer Nature Switzerland, Cham, 2022, pp. 3–14.
[8] Dunwang Qin, Zhen Peng, Lifeng Wu, Deep attention fuzzy cognitive maps for interpretable multivariate time series prediction, Knowl.-Based Syst. 275 (2023) 110700.
[9] Philippe J. Giabbanelli, Gonzalo Nápoles, Fuzzy Cognitive Maps: Best Practices and Modern Methods, Springer Nature, 2024.
[10] Yao Xixi, Ding Fengqian, Luo Chao, Time series prediction based on high-order intuitionistic fuzzy cognitive maps with variational mode decomposition, Soft Comput. 26 (2022) 189–201.
[11] Yuerong Tong, Jingyi Liu, Lina Yu, Liping Zhang, Linjun Sun, Weijun Li, Xin Ning, Jian Xu, Hong Qin, Qiang Cai, Technology investigation on time series classification and prediction, PeerJ Comput. Sci. 8 (2022).
[12] Ryan Schuerkamp, Philippe J. Giabbanelli, Extensions of fuzzy cognitive maps: A systematic review, ACM Comput. Surv. 56 (2) (2023).
[13] Sidong Xian, Xu Feng, Meerkat optimization algorithm: A new meta-heuristic optimization algorithm for solving constrained engineering problems, Expert Syst. Appl. 231 (2023) 120482.
[14] Frank Vanhoenshoven, Gonzalo Nápoles, Wojciech Froelich, Jose L. Salmeron, Koen Vanhoof, Pseudoinverse learning of Fuzzy Cognitive Maps for multivariate time series forecasting, Appl. Soft Comput. 95 (2020) 106461.
[15] Wojciech P. Hunek, Feliks Tomasz, Robust fractional-order perfect control for non-full rank plants described in the Grünwald-Letnikov IMC framework, Fract. Calc. Appl. Anal. 24 (2021) 1257–1274.
[16] Gonzalo Nápoles, Isel Grau, Agnieszka Jastrzebska, Yamisleydi Salgueiro, Long short-term cognitive networks, Neural Comput. Appl. 34 (2022) 16959–16971.
[17] Kai Wu, Kaixin Yuan, Yingzhi Teng, Jing Liu, Licheng Jiao, Broad fuzzy cognitive map systems for time series classification, Appl. Soft Comput. 128 (2022) 109458.
[18] Jun Chen, Xudong Gao, Jia Rong, Xiaoguang Gao, A situation awareness assessment method based on fuzzy cognitive maps, J. Syst. Eng. Electron. 33 (5) (2022) 1108–1122.
[19] Kai Wu, Jing Liu, Penghui Liu, Fang Shen, Online fuzzy cognitive map learning, IEEE Trans. Fuzzy Syst. 29 (7) (2021) 1885–1898.
[20] Sima Sabahi, Paul M. Stanfield, Neural network based fuzzy cognitive map, Expert Syst. Appl. 204 (2022) 117567.
[21] Georgios D. Karatzinis, Yiannis S. Boutalis, Fuzzy cognitive networks with functional weights for time series and pattern recognition applications, Appl. Soft Comput. 106 (2021) 107415.
[22] Georgios D. Karatzinis, Nikolaos A. Apostolikas, Yiannis S. Boutalis, George A. Papakostas, Fuzzy cognitive networks in diverse applications using hybrid representative structures, Int. J. Fuzzy Syst. (2023).
[23] Jingyuan Wang, Zhen Peng, Xiaoda Wang, Chao Li, Junjie Wu, Deep fuzzy cognitive maps for interpretable multivariate time series prediction, IEEE Trans. Fuzzy Syst. 29 (9) (2021) 2647–2660.
[24] Omid Orang, Petrônio Cândido de Lima e Silva, Frederico Gadelha Guimarães, Time series forecasting using fuzzy cognitive maps: a survey, Artif. Intell. Rev. 56 (2023) 7733–7794.
[25] Michal Gregor, Peter P. Groumpos, Training fuzzy cognitive maps using gradient-based supervised learning, in: Harris Papadopoulos, Andreas S. Andreou, Lazaros Iliadis, Ilias Maglogiannis (Eds.), Artificial Intelligence Applications and Innovations, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 547–556.
[26] Paul J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[27] Ikram Chraibi Kaadoud, Nicolas P. Rougier, Frédéric Alexandre, Knowledge extraction from the learning of sequences in a long short term memory (LSTM) architecture, Knowl.-Based Syst. 235 (2022) 107657.
[28] Gonzalo Nápoles, Yamisleydi Salgueiro, Isel Grau, Maikel Leon Espinosa, Recurrence-aware long-term cognitive network for explainable pattern classification, IEEE Trans. Cybern. (2022) 1–12.
[29] Isel Grau, Gonzalo Nápoles, Isis Bonet, María Matilde García, Backpropagation through time algorithm for training recurrent neural networks using variable length instances, Comput. Sist. 17 (1) (2013) 15–24.
[30] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, J. Mult.-Valued Logic Soft Comput. 17 (2–3) (2011) 255–287.
[31] Pawel Blazej, Malgorzata Wnetrzak, Pawel Mackiewicz, The role of crossover operator in evolutionary-based approach to the problem of genetic code optimization, Biosystems 150 (2016) 61–72.
[32] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, V.B. Surya Prasath, Choosing mutation and crossover ratios for genetic algorithms—A review with a new dynamic approach, Information 10 (12) (2019).