Contents lists available at ScienceDirect

Information Sciences

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

Pattern classification with Evolving Long-term Cognitive Networks

Gonzalo Nápoles [a,b,*], Agnieszka Jastrzębska [c], Yamisleydi Salgueiro [d]

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

ARTICLE INFO

Article history:
Received 18 February 2020
Received in revised form 14 August 2020
Accepted 15 August 2020
Available online 21 August 2020

Keywords:
Long-term Cognitive Networks
Recurrent neural networks
Backpropagation
Interpretability

ABSTRACT

This paper presents an interpretable neural system—termed Evolving Long-term Cognitive Network—for pattern classification. The proposed model was inspired by Fuzzy Cognitive Maps, which are interpretable recurrent neural networks for modeling and simulation. The network architecture is comprised of two neural blocks: a recurrent input layer and an output layer. The input layer is a Long-term Cognitive Network that gets unfolded in the same way as other recurrent neural networks, thus producing a sort of abstract hidden layers. In our model, we can attach meaningful linguistic labels to each neuron since the input neurons correspond to features in a given classification problem and the output neurons correspond to class labels. Moreover, we propose a variant of the backpropagation learning algorithm to compute the required parameters. This algorithm includes two new regularization components that are aimed at obtaining more interpretable knowledge representations. The numerical simulations using 58 datasets show that our model achieves higher prediction rates when compared with traditional white boxes while remaining competitive with the black boxes. Finally, we elaborate on the interpretability of our neural system using a proof of concept.

1. Introduction

Pattern classification [1] is devoted to assigning class labels to patterns based on a certain recognition model. Such a model is often trained in a supervised manner using a set of patterns whose class labels are known. Classification is one of the key machine learning tasks, so one can find a range of mature classification algorithms among which we find neural reasoning models.

Neural models are composed of neural units arranged in layers, so the task of classification is performed by forwarding the input signal through consecutive layers of the network. Forwarding is executed by multiplying the weights by the input signal encoding a certain pattern. Besides the multiplication step, there is a range of further actions to be performed with the network. Examples of such actions are the application of a transfer function to the signal, random elimination of certain values, and more as described by Huang et al. [2]. The most elemental example of a neural system is the *perceptron*. This model involves a single neuron whose actions can be broken down into three steps: multiplication (input vector by corresponding

* Corresponding author at: Department of Cognitive Science & Artificial Intelligence, Tilburg University, The Netherlands.
E-mail address: g.r.napoles@uvt.nl (G. Nápoles).

https://doi.org/10.1016/j.ins.2020.08.058
0020-0255/© 2020 The Authors. Published by Elsevier Inc.

weights), aggregation, and the application of a transfer function. As Vargas et al. [3] point out, the class label is assigned based on the value returned by the activation function. In the simplest model, the perceptron uses the unit step function which entails that it is a binary classifier. Meanwhile, as López-Rubio et al. [4] illustrate, contemporary neural models are composed of multiple layers and other transfer functions. Maji and Mullins [5] advocate to call those networks "deep" because of the increased number of hidden layers. The prevailing trend in the domain of neural models is to focus on classification accuracy with less concern about the ease of model interpretation. This happens because a complex architecture usually rewards us with high-quality predictions, but at the same time hinders the transparency of the decision mechanism attached to the model.

The class of neural models particularly relevant to the subject matter of this paper is the Recurrent Neural Network model. We depict it as a network composed of three layers: input, hidden, and output. Marquez et al. [6] conclude that the difference between the plain feedforward model and the recurrent model is that, in the feedforward model, connections are outgoing from the layer t to the layer $t+1$. Likewise, Hardy and Buonomano [7] highlight that in the recurrent model, hidden layers have connections between their neurons. These connections allow us to translate the recurrent model into an unfolded architecture. The action of unfolding replicates the hidden layer into a sequence of several hidden layers each made of the same number of neurons as the original hidden layer in the recurrent model. Wang et al. [8] mention that the unfolding is performed as many times as the designer sets. The unfolded architecture preserves the basic properties of the feedforward model, namely, there are only connections from the layer t to $t+1$ and there are no connections the other way around. Liu and collaborators emphasize that Recurrent Neural Networks are usually deep because the unfolding adds hidden layers [9]. However, as stressed by Han et al. [10], the interpretation of weights between hidden layers, similarly as the interpretation of weights in other deep neural models, is not feasible. This happens because hidden neurons lack meaning.

When it comes to building naturally interpretable neural systems (i.e., models that allow explaining their reasoning process by themselves), Fuzzy Cognitive Maps (FCMs) [11] provide some interesting characteristics. Froelich and Pedrycz [12] described them as information networks which are composed of neural concepts (which are equivalent to neurons in traditional neural networks) and causal relationships. An important feature of FCM-based models is that each concept corresponds with a phenomenon, variable, problem feature or entity, while graph arcs denote the relationship between the concepts. This means that every component in these recurrent neural systems involves a well-defined meaning for the problem being modeled. Fig. 1 shows a generic FCM-based model incoming three neural concepts with self-connections.

In FCMs, arcs are weighted with real numbers in the $[-1,1]$ interval, that is $w_{ji} \in [-1,1]$ for $j,i = 1,\dots,P$ where P is the number of neurons in the network. In the model depicted in Fig. 1, $a_1^{(t)}, a_2^{(t)}, \dots, a_P^{(t)}$ denote the activation values of neurons in the t-th iteration, which describe the current state of a given phenomenon. During the reasoning, the FCM processes this state and produces $a^{(t+1)}$ and so on, until a stop condition is met.

Unfortunately, FCMs are not very good at solving classification problems. There are two major reasons for this. Firstly, as Nápoles et al. [13] point out, the weights are confined to the $[-1,1]$ interval, thus making FCMs' prediction horizon quite limited when compared to other neural models. Secondly, we can indeed decompose any FCM into a deep feed-forward neural network, however, its width will be determined by the number of neural concepts. On top of that, the weights connecting these abstract layers will not change from an iteration to another. Aiming at overcoming the first problem, Nápoles et al. [13] 14 introduced the Short-term Cognitive Networks (STCNs) and the Long-term Cognitive Networks (LTCNs), respectively. However, the problem of having fixed weights from an iteration to another persists.

In this paper, we present an interpretable neural system—termed *Evolving Long-term Cognitive Network* (ELTCN)— for pattern classification. The ELTCN model builds upon the LTCNs developed by Nápoles et al. [13], which provide interesting interpretability features. The distinctive characteristic of this model is that it allows for the LTCN's weights to change from an iteration to another during the reasoning process. We envisioned that the LTCN model can be fused as the input layer of our neural classifier, while its outputs can be connected to the decision neurons. This model can get unfolded without losing the ability to interpret the neurons and weights. The proposed construction methodology aims at foregoing the standard perception of neural networks as black boxes. Having well-defined neurons facilitates immediate understanding of the model and the decision process it illustrates, however, the lack of hidden neurons might hinder its accuracy. Hence, the second contribution of this paper is related to a backpropagation algorithm which is used to adjust the weights and some transfer func-

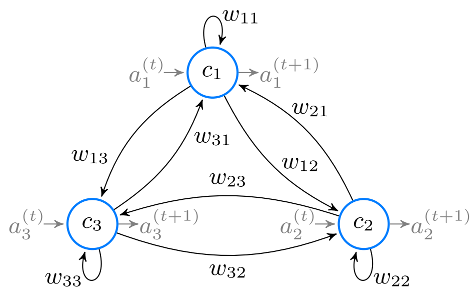

Fig. 1. A generic Fuzzy Cognitive Map with three nodes.

tion parameters. This learning algorithm includes two new regularization components that attempt at producing networks with little variability in their dynamic parameters. Finally, we explain how to get insight into the decision mechanism of the network by determining the importance of each neural concept in the classification process.

The rest of this paper is organized as follows. Section 2 revises the existing approaches to pattern classification based on FCMs. Section 3 describes the theoretical underpinnings of the LTCN model. Section 4 presents our neural system, whereas Section 5 elaborates on the backpropagation algorithm and the regularization components. Section 6 describes a procedure to normalize the weights attached to the network, which allows better inspecting the model. Section 7 contains an empirical analysis that covers both the performance and interpretability angles. Section 8 concludes the paper.

2. Pattern classification with Fuzzy Cognitive Maps

Our proposal falls into the category of Cognitive Map-based models, which have been used to solve pattern classification problems. Next, we will discuss the most prominent approaches reported in the literature.

Firstly, it shall be mentioned that the majority of research on classification with Cognitive Maps is devoted to the application of the FCM model that operates on the problem features directly. Other kinds of Cognitive Maps are rarely applied. For example, Nápoles et al. [15] used Granular Cognitive Maps, which are a generalization of the FCM model.

The existing methodology for pattern classification with FCMs is based on the idea of using separate neural concepts that return the most likely decision class. Typically, there are N of such decision concepts, where N is the number of classes in a given multi-class classification problem. Papageorgiou and Kannappan [16] explored this approach. In contrast, Natarajan et al. [17] discussed a model with a single decision neuron where the activation values are discretized to obtain class labels. If we would switch to the terminology used in the neural network literature, then the concepts performing the class assignment would form the final (output) layer. The first layer (input layer) is made of concepts corresponding to features describing the patterns. Sometimes, the architecture is just made of those two layers. Christodoulou et al. [18] followed this approach, while in the work of Song et al. [19], the FCM is constructed with one or more intermediate layers, which would be the hidden layers in the neural network formalism. Imposing the layer-like architecture upon an FCM-based model entails the necessity of canceling certain weights. In particular, connections originating from the output layer and incoming to the input layer are removed. In most cases, with a few exceptions like the ones discussed by Guo et al. [20], links between output concepts are removed as well. In another study, Nápoles et al. [21] suggested suppressing self-loops.

An important topic when building FCM-based classifiers refers to the learning algorithm. This aspect is of elemental importance since the weight matrix determines the quality of classification. A popular approach researched by Papakostas et al. [22] is the Hebbian-like learning. It starts with an initial weight matrix that should ideally be given by the experts. The algorithm performs an iterative routine that calculates activation values in the next iteration based on current activation values and updates the weights. The new weights are typically computed as a difference between the current weights and some small value, which is obtained by multiplying a learning rate by some expression involving the difference between the computed and expected activation values. Some algorithms, such as the Differential Hebbian Learning studied by Salmeron and Palos-Sanchez [23], do not perform well when the initial state is random. It shall be also mentioned that Amirkhani et al. [24] utilized Active Hebbian Learning for FCM-based classifiers. Active Hebbian Learning is also present in the work of Papakostas et al. [22], where six different types of Hebbian learning algorithms were tested. The conclusion of this study is that such algorithms perform poorly. Overall, we strongly suggest discontinuing the usage of Hebbian-type algorithms for training FCM-based prediction models.

Heuristic optimization strategies have also been researched. For example, Kannappan and Papageorgiou [25] explored the application of Adaptive Clonal Selection in a map-based classification procedure. Heuristic search methods modify one or several candidate solutions in each iteration, according to some predefined update formula. While these methods often provide better prediction rates than Hebbian-like approaches, they have several drawbacks that must be taken into consideration. For example, they are slow and their overall performance is often compromised as non-progress situations (e.g., stagnation, premature convergence) lead to very poor local optima.

Bhutani et al. [26] presented an FCM-based classifier whose processing scheme is entirely different from the models discussed so far. This model uses two layers. The first layer is interpreted as the features' domain and the output layer is interpreted as class labels. The FCM was constructed based on an already-existing fuzzy inference model. Therefore, we can say that the actual usage of the FCM was only to illustrate the existing inference model. Pajares et al. [27] investigated another, entirely distinct usage of FCMs for pattern classification. In this research, the authors designed a two-step procedure for pixel clustering and then image classification. They proposed to create k FCM models, where k is the number of desired clusters, such that each concept denotes a pixel in the original image. After performing several iterations of the FCM reasoning, the outputs of the FCMs were interpreted as degrees of membership to the decision classes. Despite the novelty, image classification is something that can be successfully done with existing deep learning methods. Thus, in terms of accuracy, an FCM-based classifier would report little added value.

What the literature review shows is that there is a lack of FCM-based classifiers equipped with learning algorithms having strong mathematical foundations. As Nair et al. [28] underline, such algorithms must take into consideration the main advantage of FCM-based classifiers: they can *naturally* elucidate their decision process without the need of using a post-hoc procedure. On one hand, the FCM-based classifiers should produce competitive prediction rates when compared with traditional classifiers. On the other hand, the interpretability feature of FCMs has been reduced to showing the learned weights to domain experts such that they can draw conclusions by themselves. However, the extent to which human beings can understand the decision process of a network comprised of hundreds of weights is questionable.

In the next section, we describe the theoretical foundations behind the LTCN model, which is deemed the cornerstone of our proposal.

3. Long-term Cognitive Networks

Roughly speaking, LTCNs are recurrent neural networks where each neural concept c_i denotes either an input or output (continuous) variable in a given domain. In these interpretable neural systems, the weight $w_{ji} \in \mathbb{R}$ denotes the rate of change in the conditional mean of c_i with respect to c_j, assuming that the activation values of the remaining neurons impacting c_i are fixed. This is similar to interpreting the coefficients in a (logistic) regression model. Section 7 will further elaborate on this feature. Nápoles et al. [14] point out that hidden neurons are not allowed as they do not correspond with any problem feature. The weights in LTCNs follow the rules depicted below:

- $w_{ji} > 0$: If positively activated, c_j will impact positively c_i. Thus, the higher (lower) the activation value of c_j in the current iteration, the higher (lower) the value of c_i in the following iteration;
- $w_{ji} < 0$: If positively activated, c_j will impact negatively c_i. Thus, the higher (lower) the activation value of c_j in the current iteration, the lower (higher) the value of c_i in the following iteration.

Eq. (1) shows the reasoning rule of LTCNs, which computes the activation value $a_i^{(t+1)}$ of the neural concept c_i in the $(t+1)$-th iteration for a given input pattern as the initial activation vector,

$$a_i^{(t+1)} = f_i^{(t+1)}\left(\sum_{j=1}^{P} w_{ji} a_j^{(t)}\right) \tag{1}$$

where P is the number of neural concepts in the network, whereas $f_i^{(t+1)}(\cdot)$ is the transfer function adopted to confine the activation value of each neuron to the desired interval. The reader can notice that the transfer function attached to each neural concept can change from one iteration to another. The nonsynaptic learning of LTCNs proposed by Nápoles et al. [13] [29] is devoted to adjusting the shape of each transfer function in each iteration while preserving the weights defined by domain experts. However, in the neural system depicted in the next section, only the weights and the function offset (which is equivalent to the bias component in other neural models) will be adjusted. The remaining transfer function parameters will hold for all neural concepts and iterations, thus, the nonsynaptic learning is no longer required.

4. Evolving Long-term Cognitive Networks

In this section, we present a new LTCN-based neural system termed *Evolving Long-term Cognitive Networks* for pattern classification.

4.1. Network architecture

Duda et al. [1] describe the supervised classification problem as the task of building a mapping $\Gamma : X \rightarrow Y$ that assigns to each instance $x \in X$ the proper decision class $y \in Y$ such that $Y = \{y_1, y_2, \ldots, y_N\}$. Each problem instance x is described by a set of variables or features $x_1, \ldots, x_M$. Therefore, we have M input variables and N possible outcomes. This problem can be modeled by using an LTCN-based architecture. Before going any further, it is worth reiterating that the added value of our proposal would rely on its interpretability rather than obtaining outstanding prediction rates.

As explained by Nápoles et al. [21], problem variables are mapped onto input neural concepts, which can be either dependent or independent. The former ones can be influenced by other input neurons, while the latter ones just propagate their initial activation vector without being influenced by any other input neuron, therefore, their activation values remain static. Output neurons are used to compute the decision class for an initial activation vector.

In our neural system, each input neuron is connected with the others (unless the experts state otherwise) thus providing the system with a recurrent network topology. Moreover, each input neuron is connected with the outputs. Fig. 2 displays, as an example, the network topology for a classification problem with three features and three decision classes.

4.2. Evolving reasoning rule

One might think that this type of neural architecture is “limited” when it comes to the number of hidden neurons and layers. On one hand, the *universal approximation theorem* (as discussed by Leshno et al. [30] and Scarselli and Chung Tsoi [31]) is clear about the role of hidden neurons to compute good approximations. On the other hand, we should take into account that any recurrent neural network can be unfolded into a multilayer network with a fixed width but unlimited

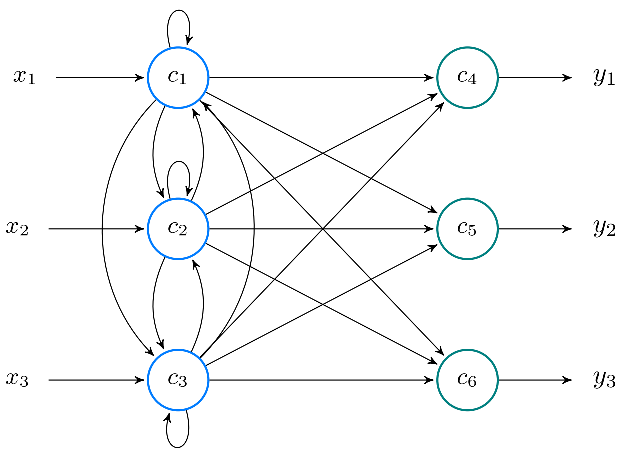

Fig. 2. Neural model comprised of $M = 3$ input neurons (c_1, c_2, c_3) and $N = 3$ output neurons (c_4, c_5, c_6). Overall, the network has $P = M + N = 6$ neurons. In this example, the input signal is denoted as x_i while output signal is denoted as y_i. These two values correspond to $a_i^{(0)}$ and $a_i^{(T)}$, respectively.

length, theoretically speaking. The problem with this multilayer network is that we will have the same weight connecting the neurons c_i and c_j in all abstract hidden layers.

Aiming at overcoming this issue, we will allow our neural system to change its weights from an iteration to another as it performs the recurrent reasoning process. This causes the emergence of T abstract hidden layers, each containing M abstract hidden neurons, thus equipping the network with improved learning capabilities. Eq. (2) shows how to compute neurons' activation values by following the evolving reasoning principle,

$$a_i^{(t+1)} = f_i^{(t+1)}\left(\sum_{j=1}^{P} w_{ji}^{(t)} a_j^{(t)}\right) \tag{2}$$

where $f_i^{(t+1)}(x)$ can be either the sigmoid function,

$$s_i^{(t)}(x) = \frac{1}{1 + e^{-\lambda_i^{(t)}(x - h_i^{(t)})}} \tag{3}$$

or the hyperbolic tangent function,

$$q_i^{(t)}(x) = \frac{e^{2\lambda_i^{(t)}(x - h_i^{(t)})} - 1}{e^{2\lambda_i^{(t)}(x - h_i^{(t)})} + 1} \tag{4}$$

such that $\lambda_i^{(t)} > 0$ and $h_i^{(t)} \in \mathbb{R}$ are two parameters denoting the function slope and its offset, respectively. Given N decision classes, the activation values for output neurons will be computed as follows:

$$a_i^{(t+1)} = \frac{e^{\left(\sum_{j=1}^{M} w_{ji}^{(t)} a_j^{(t)}\right)}}{e^{\left(\sum_{k=1}^{N}\left(\sum_{j=1}^{M} w_{jk}^{(t)} a_j^{(t)}\right)\right)}}. \tag{5}$$

It is worth highlighting that, unlike LTCNs, the proposed ELTCN model does not aim at preserving all weights defined by the expert while focusing on learning the nonsynaptic parameters. Overall, we need parameters having two main properties: 1) they produce competitive prediction rates when compared with state-of-the-art classifiers, 2) they can elucidate, to the same extent, how the model arrived at a particular decision.

5. Supervised learning

This section explains how to train the proposed neural system. Firstly, we introduce a backpropagation algorithm to estimate the network parameters (i.e., the weights and the transfer function offsets). Secondly, some of these equations are rewritten to include two regularization components, which allow minimizing the weight variability between two consecutive abstract layers (thus making the model easier to interpret) and the offset values.

5.1. Backpropagation algorithm

The backpropagation algorithm is devoted to estimating the weights $w_{ji}^{(t)}$ and the offset parameter $h_i^{(t)}$ associated with the i-th neuron in each iteration. Notice that optimizing the slope $\lambda_i^{(t)}$ does not bring much-added value since it would only rescale the argument of each transfer function. Instead, this parameter will be used to normalize the weights such that we can produce weights in the $[-1,1]$ interval, which is often appreciated by the experts.

As a first step, we perform the forward pass to compute the total error $\mathcal{E}$, which is calculated as the cross-entropy between the expected response vector Y and the predicted one for a given instance.

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

$$\mathcal{E} = -\sum_{i=1}^{N} y_i log(a_i^{(t)}) \tag{6}$$

where y_i is the actual value of the i-th decision class, while $a_i^{(t)}$ is the activation value of that decision neuron. Consequently, we have:

$$\frac{\partial \mathcal{E}}{\partial a_i^{(t)}} = -\frac{y_i}{a_i^{(t)}} + \frac{1-y_i}{1-a_i^{(t)}}. \tag{7}$$

The output neurons use a *softmax* transfer function. Therefore, we need to calculate the partial derivatives of the output they produce $a_i^{(t)}$ with respect to the raw activation values $\bar{a}_j^{(t)}$. More generally, we have:

$$\frac{\partial a_i^{(t)}}{\bar{a}_j^{(t)}} = \begin{cases} \bar{a}_i^{(t)}(1-\bar{a}_j^{(t)}) & i = j \\ -\bar{a}_i^{(t)}\bar{a}_j^{(t)} & i \neq j \end{cases}. \tag{8}$$

Case 2. When $1 < t < T$, we have the following:

$$\frac{\partial \mathcal{E}}{\partial a_i^{(t)}} = \sum_{j=1}^{M} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}} \times \frac{\partial a_j^{(t+1)}}{\partial a_i^{(t)}} = \sum_{j=1}^{M} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}} \times \frac{\partial a_j^{(t+1)}}{\partial \bar{a}_j^{(t+1)}} \times \frac{\partial \bar{a}_j^{(t+1)}}{\partial a_i^{(t)}} = \sum_{j=1}^{M} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}} \times \frac{\partial a_j^{(t+1)}}{\partial \bar{a}_j^{(t+1)}} \times w_{ij} \tag{9}$$

where $\frac{\partial a_j^{(t+1)}}{\partial \bar{a}_j^{(t+1)}} = f_j'^{(t+1)}(x)$. For the sigmoid function, we have:

$$\frac{\partial a_j^{(t+1)}}{\partial \bar{a}_j^{(t+1)}} = \frac{1}{4}\lambda_j^{(t+1)}\text{sech}^2\left(\frac{1}{2}\lambda_j^{(t+1)}\left(\bar{a}_j^{(t+1)} - h_j^{(t+1)}\right)\right)$$

while for the hyperbolic tangent, we have:

$$\frac{\partial a_j^{(t+1)}}{\partial \bar{a}_j^{(t+1)}} = \lambda_j^{(t+1)}\text{sech}^2\left(\lambda_j^{(t+1)}\left(\bar{a}_j^{(t+1)} - h_j^{(t+1)}\right)\right).$$

Next, we need to obtain the partial derivatives of the total error with respect to the transfer function parameters $h_i^{(t)}$ as follows:

$$\frac{\partial \mathcal{E}}{\partial h_i^{(t)}} = \frac{\partial \mathcal{E}}{\partial a_i^{(t)}} \times \frac{\partial a_i^{(t)}}{\partial h_i^{(t)}}. \tag{10}$$

- For the sigmoid transfer function, we have:

$$\frac{\partial a_i^{(t)}}{\partial h_i^{(t)}} = -\frac{1}{4}\lambda_i^{(t)}\text{sech}^2\left(\frac{1}{2}\lambda_i^{(t)}\left(\bar{a}_i^{(t)} - h_i^{(t)}\right)\right). \tag{11}$$

- For the hyperbolic tangent function, we have:

$$\frac{\partial a_i^{(t)}}{\partial h_i^{(t)}} = \lambda_i^{(t)}\left(-\text{sech}^2\left(\lambda_i^{(t)}\left(\bar{a}_i^{(t)} - h_i^{(t)}\right)\right)\right). \tag{12}$$

Eq. (13) shows how to compute the partial derivative of the total error with respect to the weights in the t-th abstract layer,

$$\frac{\partial \mathcal{E}}{\partial w_{ij}^{(t)}} = \frac{\partial \mathcal{E}}{\partial a_j^{(t)}} \times \frac{\partial a_j^{(t)}}{\partial \bar{a}_j^{(t)}} \times \frac{\partial \bar{a}_j^{(t)}}{\partial w_{ij}^{(t)}} \tag{13}$$

such that

$$\frac{\partial \bar{a}_j^{(t)}}{\partial w_{ij}^{(t)}} = \frac{\partial \left(\sum_{l=1}^{M} a_i^{(t-1)} w_{lj}^{(t)} \right)}{\partial w_{ij}^{(t)}} = a_i^{(t-1)}. \tag{14}$$

Eq. (15) shows the gradient vector for the transfer function parameters attached to the (input) neurons in the t-th abstract layer,

$$\nabla_h^{(t)} \mathcal{E} = \left(\frac{\partial \mathcal{E}}{\partial h_1^{(t)}}, \ldots, \frac{\partial \mathcal{E}}{\partial h_i^{(t)}}, \ldots, \frac{\partial \mathcal{E}}{\partial h_M^{(t)}} \right). \tag{15}$$

Similarly, Eq. (16) shows the gradient vector corresponding with the weights connecting the t-th abstract layer with the following one,

$$\nabla_w^{(t)} \mathcal{E} = \left(\frac{\partial \mathcal{E}}{\partial w_{11}^{(t)}}, \ldots, \frac{\partial \mathcal{E}}{\partial w_{ij}^{(t)}}, \ldots, \frac{\partial \mathcal{E}}{\partial w_{MM}^{(t)}} \right). \tag{16}$$

Notice that these gradient vectors are all we need to adjust the network parameters employing a gradient descent method. In this paper, the Adam optimization method [32]33 will be used during the numerical simulations. It is worth mentioning that the proposed backpropagation algorithm could be adjusted to deal with a large number of abstract layers (recurrent iterations), however, that is beyond the scope of this paper.

5.2. Regularization

The reader can notice that the backpropagation algorithm presented in the previous section aims at obtaining the highest prediction rates possible, so it does not differ much from the traditional approach. Particularly, the loss function does not take into consideration the network variability along the recurrent process. This might result in "unsteady" models with a significant variability of the weights connecting two consecutive abstract layers. That would hinder the process of deriving consistent explanations.

To deal with this problem, we introduce two new regularization methods. As Kim et al. [34] underline, regularization itself is a technique frequently applied in neural models to reduce potential overfitting. However, our regularization methods are not oriented to improving the generalization capabilities of ELTCN-based classifiers. Instead, they attempt to reduce the variability of weights from an abstract layer to another and the offset parameters attached to the transfer functions. The former strategy allows obtaining more consistent weights through the ELTCN's reasoning process while the latter reduces the impact of the bias components on the classification process.

In order to introduce the new regularization components, we have to modify some equations of the backpropagation method presented in the previous subsection. The first modification concerns the loss (error) function in Eq. (6) that computes the global dissimilarity between the actual values stored in the dataset and the responses produced by our neural system.

Eq. (17) shows the new error function (that replaces the Eq. (6)) which includes the regularization components,

$$\mathcal{E} = -\sum_{i=1}^{N} y_i log(a_i^{(t)}) + \frac{1}{2}(\mathcal{R}_1 + \mathcal{R}_2) \tag{17}$$

such that

$$\mathcal{R}_1 = \sum_{t=1}^{T-1} \sum_{i=1}^{M} \sum_{j=1}^{M} \beta_{ij} \left(\Delta w_{ij}^{(t)} \right)^2 \tag{18}$$

where $\beta_{ij} \geqslant 0$ is a user-specified parameter to control the extent to which we allow the weight w_{ij} to change from the current iteration to the next one, $\Delta w_{ij}^{(t)} = w_{ij}^{(t)} - w_{ij}^{(t+1)}$, while $\mathcal{R}_2$ is defined as follows:

$$\mathcal{R}_2 = \sum_{t=1}^{T} \sum_{i=1}^{M} \psi_i \left(h_i^{(t)} \right)^2 \tag{19}$$

such that $\psi_i \geqslant 0$ is another user-specified parameter to control the extent to which we allow the offset parameter $h_i^{(t)}$ to increase. Larger parameter values cause the error function in Eq. (17) to be more harshly penalized when the absolute values of the offset parameters increase.

The second modification to the backpropagation algorithm concerns the gradient equations used to update the weights and the offset parameters (Eq. (15) and Eq. (16), respectively). This needs to be done with caution. For example, when using stochastic gradient descent, the L_2 regularization and *weight decay regularization* [35] are equivalent. This is not necessarily true for all gradient-based learning algorithms. Loshchilov and Hutter [33] reported that, when combined with adaptive gradients, the L_2 regularization leads to weights with large historic parameter and/or gradient amplitudes being regularized less than they would have been when using weight decay.

Having mentioned this issue, we introduce Eqs. (20) and (21) to replace Eq. (10) and Eq. (13), respectively,

$$\frac{\partial \mathcal{E}}{\partial h_i^{(t)}} = \frac{\partial \mathcal{E}}{\partial a_i^{(t)}} \times \frac{\partial a_i^{(t)}}{\partial h_i^{(t)}} + \lambda_i h_i^{(t)}, \tag{20}$$

$$\frac{\partial \mathcal{E}}{\partial w_{ij}^{(t)}} = \frac{\partial \mathcal{E}}{\partial a_j^{(t)}} \times \frac{\partial a_j^{(t)}}{\partial \bar{a}_j^{(t)}} \times \frac{\partial \bar{a}_j^{(t)}}{\partial w_{ij}^{(t)}} + \beta_{ij} \Delta w_{ij}^{(t)}. \tag{21}$$

In summary, the regularization term $\mathcal{R}_1$ attempts to reduce the variability between two consecutive weight matrices, thus forcing the model to produce "stable" weight sets across the evolving process. On the other hand, regularization term $\mathcal{R}_2$ aims to reduce the L_2-norm of the offset vector at each iteration, therefore the predictions will mainly rely on the weights. The reader can notice that both regularization terms have the common purpose of producing neural models being easier to interpret.

Algorithm 1 presents the steps needed to build and train our neural classifier. Overall, we need the training set $Z = [X|Y]$, where X is a matrix containing the initial activation vectors for the input neurons and Y contains the corresponding outputs, the number of abstract layers T, the number of training epochs Q and the regularization parameters β_{ij} and ψ_i.

Algorithm 1: Construction and learning of ELTCN classifiers.

```
Input: training set $Z$, layers $T$, parameters $\beta_{ij}$ and $\psi_i$, epochs $Q$
Output: trained ELTCN model
1  #construction process
2  Generate the network's architecture: $\mathbf{M}$ input neurons, $\mathbf{T}$ abstract
     hidden layers, and $\mathbf{N}$ output neurons
3  #training process
4  foreach epoch in $Q$ do
5      foreach batch in $Z$ do
6          Perform forward pass all instances in batch
7          Compute total error $\mathcal{E}$ by Equation (17)
8          foreach $t$ in $T$ do
9              Compute the partial derivatives of $\mathcal{E}$ with respect to $h_i^{(t)}$ by
                 Equation (20)
10             Compute the partial derivative of $\mathcal{E}$ with respect to $w_{ij}^{(t)}$ by
                 Equation (21)
11             Update bias $h_i^{(t)}$ and weights $w_{ij}^{(t)}$, respectively
```

Notice that the number of input neurons M and the number of outputs N can be obtained directly from the training set Z. Algorithm 1 does not include the hyperparameters associated with the optimization algorithm since they often change from an optimizer to another.

6. Network normalization

After the learning phase is complete, we can interpret the knowledge learned from historical data. This can be achieved by either getting insight into how the network arrived at the decision class given a specific instance or inspecting the network as a whole. In the former case, we would need the final activation values of neurons and the weight matrix, while in the latter case only the weight matrix would suffice. Whichever the case might be, we can certainly increase the transparency of our model by transforming the weights into the $[-1, 1]$ interval. Recently, Nápoles et al. [36] introduced a weight normalization procedure that can be adapted to the proposed model.

Let $W_I^{(t)}$ and W_O be the weight matrices connecting the input neurons in the t-th abstract layer, and the input neurons with the output ones, respectively. Furthermore, let $a_i^{(t+1)}$ be the value produced by the i-th neural concept in the $(t+1)$-th iteration, which is computed as follows:

$$a_i^{(t+1)} = f_i^{(t+1)}\left(\bar{a}_i^{(t+1)}\right) \tag{22}$$

where

$$\bar{a}_i^{(t+1)} = -\lambda_i(w_{1i}a_1^{(t)} + \ldots + w_{ji}a_j^{(t)} + \ldots + w_{Mi}a_M^{(t)} - h_i).$$

Aiming at normalizing the weights in $W_I^{(t)}$ and W_O, we should determine whether c_i is an input neuron or a decision one. For example, if c_i is an input neuron then we can rewrite $\bar{a}_i^{(t+1)}$ as follows:

$$\bar{a}_i^{(t+1)} = -\lambda_i\phi_i\left(\frac{w_{1i}}{\phi_i}a_1^{(t)} + \ldots + \frac{w_{ji}}{\phi_i}a_j^{(t)} + \ldots + \frac{w_{Mi}}{\phi_i}a_M^{(t)} - \frac{h_i}{\phi_i}\right)$$

such that $\phi_i = max_j\{|w_{ji}^{(t)}|\}, \forall t$ and $w_{ji} \in W_I^{(t)}$. If we now make $w'_{ji} = \frac{w_{ji}}{\phi_i}, \lambda'_i = \lambda_i\phi_i$ and $h'_i = \frac{h_i}{\phi_i}$ then we obtain:

$$\bar{a}_i^{(t+1)} = -\lambda'_i(w'_{1i}a_1^{(t)} + \ldots + w'_{ji}a_j^{(t)} + \ldots + w'_{Mi}a_M^{(t)} - h'_i)$$

such that the weight $w'_{ji} \in [-1, 1]$ as desired. The same procedure can be applied to the weights connecting the input neurons with the output ones, which would imply that $\phi_i = max_j\{|w_{ji}^{(t)}|\}, \forall t$ and $w_{ji} \in W_O$.

The reader might wonder why the weight normalization is performed separately when we could simply make $\phi_i = max_j\{|w_{ji}^{(t)}|\}, \forall t$ and $w_{ji} \in (W_I^{(t)}|W_O)$. While that would be a possibility indeed, we have to take into consideration that these matrices play different roles within our neural system, hence it would be reasonable to assume that they behave differently.

7. Numerical simulations

In this section, we explore both the predictive power of the ELTCN model and its interpretability. The steps of the experimental methodology can be summarized as follows: (i) select representative datasets devoted to traditional pattern classification, (ii) explore the effect of regularization parameters on algorithm's behavior, (iii) choose the state-of-the-art classifiers to be used for comparison, (iv) select the performance metric and conduct the statistical analysis, and (v) illustrate how to derive explanations from the knowledge structures attached to the network. All the details of each step are given in the following subsections together with the research hypotheses.

7.1. Benchmark problems

The experiments concerned 58 pattern classification datasets. Such benchmark problems (see Table 1) have been collected both from the KEEL [37] and the UCI Machine Learning [38] repositories. Benchmark problems emulate different challenges that may occur in real-life data and allow assessing the classifier's performance on different scenarios.

A closer inspection of these datasets shows that the number of instances goes from 106 to 10,992, the number of features ranges from 3 to 262 while the number of classes goes from 2 to 100. In this pool, we have 10 noisy datasets and 18 imbalanced datasets (we say that a dataset is imbalanced if the ratio between the majority and minority class sizes exceeds 5:1). During the data preparation step, we have normalized the numerical features, removed identifier-type ones, and replaced the missing values with the mean or the mode depending on the feature being numerical or nominal.

7.2. Exploring the effect of regularization parameters

As mentioned in Section 5.2, the proposed learning algorithm includes the regularization components $\mathcal{R}_1$ and $\mathcal{R}_2$. The former (see Eq. (18)) includes a user-specified parameter β_{ij} that controls the variability of w_{ji} from one iteration to another. The latter (see Eq. (19)) involves the parameter ψ_i to control the size of the transfer function offset. In this subsection, we explore the sensitivity of the proposed model to both parameters.

Table 1
Benchmark problems used during the simulations.

ID	Name	Instances	Features	Classes	Noisy	Imbalance
1	acute-inflammation	120	6	2	no	no
2	acute-nephritis	120	6	2	no	no
3	appendicitis	106	7	2	no	no
4	arrhythmia	452	262	13	no	123:1
5	balance-noise	625	4	3	yes	5:1
6	balance-scale	625	4	3	no	5:1
7	blood	748	4	2	no	no
8	breast-cancer-wisc-prog	198	34	2	no	no
9	cardiotocography-10	2,126	19	10	no	11:1
10	cardiotocography-3	2,126	35	3	no	10:1
11	collins	500	21	15	no	13:1
12	dermatology	366	15	6	no	6:1
13	ecoli	336	7	8	no	71:1
14	flags	194	28	8	no	15:1
15	glass	214	9	6	no	8:1
16	haberman	306	3	2	no	no
17	heart-5an-nn	270	13	2	yes	no
18	heart-statlog	270	13	2	no	no
19	ionosphere	351	34	2	no	no
20	iris	150	4	3	no	no
21	iris-10an-nn	150	4	3	yes	no
22	iris-20an-nn	150	4	3	yes	no
23	iris-5an-nn	150	4	3	yes	no
24	libras	360	26	15	no	no
25	liver-disorders	345	6	2	no	no
26	mfeat-morphological	2,000	6	10	no	no
27	mfeat-zernike	2,000	25	10	no	no
28	monk-2	432	6	2	no	no
29	new-thyroid	215	5	2	no	5:1
30	optdigits	5,620	64	10	no	no
31	ozone	2,536	72	2	no	34:1
32	page-blocks	5,473	10	5	no	175:1
33	parkinsons	195	22	2	no	no
34	pendigits	10,992	13	10	no	no
35	pima	768	8	2	no	no
36	pima-10an-nn	768	8	2	yes	no
37	pima-20an-nn	768	8	2	yes	no
38	pima-5an-nn	768	8	2	yes	no
39	planning-relax	182	12	2	no	no
40	plant-shape	1,600	64	100	no	no
41	saheart	462	9	2	no	no
42	segment	2,301	19	7	no	no
43	sonar	208	60	2	no	no
44	spectrometer-1	531	100	48	no	55:1
45	vehicle	846	18	4	no	no
46	vehicle0	846	18	2	no	no
47	vehicle1	846	18	2	no	no
48	vehicle2	846	18	2	no	no
49	vertebra-column-2c	310	6	2	no	no
50	vertebra-column-3c	310	6	3	no	no
51	wall-following	5,456	24	4	no	7:1
52	waveform	5,000	40	3	no	no
53	wine	178	13	3	no	no
54	wine-10an-nn	178	13	3	yes	no
55	wine-5an-nn	178	13	3	yes	no
56	winequality-red	1,599	11	6	no	68:1
57	winequality-white	4,898	11	7	no	440:1
58	yeast	1,484	8	10	no	93:1

In this following experiments, the ELTCN classifier was configured as follows. We used the popular Adam optimization algorithm [32] with its default parameter setting, the number of epochs was set to 200, the number of abstract layers was set to five, and the number of folds in the cross-validation process was set to five to keep the simulations time manageable. The number of folds will be increased to 10 when comparing our model against state-of-the-art classifiers, which is a more popular setting in the literature.

The first measure to be explored in this analysis concerns the average Kappa statistic for different parameter settings. While accuracy is considered a mainstream measure of classification quality, the Kappa statistics is a more robust measure.

This coefficient takes into account the agreement occurring by chance, which Japkowicz and Shah [39] report to be especially relevant for datasets with class imbalance. The second measure is the average absolute variability of weights across all abstract layers. Overall, we are interested in measuring the effect of the regularization parameters on both the accuracy and the variability attached to the proposed neural classifier.

Fig. 3 shows the average Kappa and the average absolute variability of weights when changing the values of β_{ij} and ψ_i. The average values are computed over the 58 datasets using a 5-fold cross-validation process. Moreover, the parameter values change from 0 to 0.5 with a step size of 0.01. The values of β_{ij} and ψ_i change evenly for all components.

Fig. 3 a) shows that the performance goes down when increasing the values of both regularization parameters. Similarly, in Fig. 3 b), we can observe a tendency to decrease the variability of the weights as both parameters increase. It can also be noticed that the model is more sensitive to the β_{ij} parameter since it controls the extent to which the weights can differ for consecutive abstract layers. Overall, we suggest using small positive values to configure β_{ij} and ψ_i to preserve the accuracy as much as possible. Furthermore, we can combine the insights derived from this simulation with available domain knowledge or data-driven heuristics. For example, we can use the correlation between the problem variables as illustrated in the following subsection.

7.3. Methodology and state-of-the-art algorithms

In this subsection, we introduce the traditional models used for comparison and discuss the key hypotheses to be explored.

The state-of-the-art classifiers selected for comparison can be split into two groups: white boxes and black boxes. The former ones are deemed interpretable (with varying extents) but less accurate when compared with black boxes. Within the first group, we have Logistic Regression (LR), Gaussian Naive Bayes (GNB), and Decision Trees (DT). Representatives of the second group include Support Vector Machines (SVM), Random Forests (RF) and Multilayer Perceptron (MLP) using two hidden layers.

Similarly to the proposed ELTCN model, the MLP classifier uses the Adam optimization algorithm and the sparse cross-entropy to measure the error during the training process. The numbers of epochs in Adam was set to 200 in all cases to keep the simulation time low. Table 2 summarizes the parameter settings of the mentioned classifiers. The remaining parameters were set to default values as reported in the *Scikit-learn* package.

In our experiments, $\beta_{ij} = 0.01$ in Eq. (18) if the absolute correlation between variables x_i and x_j is above 0.5, otherwise $\beta_{ij} = 0$. It should be stated, that this heuristic strategy seeks to reduce the variability of weights that connect highly correlated variables as it would be reasonable to assume that the related neurons are good estimators. Other strategies oriented to injecting knowledge into the system are also possible, which is considered a strength of our proposal. Likewise, we assume that $\psi_i = 0.01$ in Eq. (19). Overall, the $\mathcal{R}_1$ regularizer seeks to minimize the variability of interesting weights while the $\mathcal{R}_2$ regularizer attempts to reduce the offset values as much as possible, such that the classification relies on the weights. Finally, our model performs $T = 5$ iterations during the recurrent inference process.

Notice that the envisaged simulations do not include a hyperparameter tuning step. This comes with advantages and disadvantages. On one hand, optimizing the hyperparameters allows determining the optimal prediction performance of each model in each dataset. However, this increases the computational difficulty of building the models while also demanding more data to create the validation sets. On the other hand, using default values allows testing the model's performance in different situations without investing time in producing optimal values. The disadvantage of this strategy is that we would not know the extent to which the model can produce better results. In reality, both approaches provide two different, equally

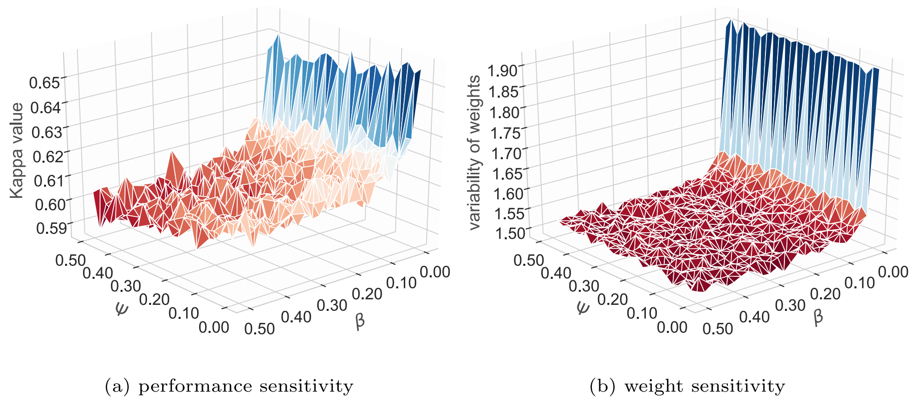

Fig. 3. Effect of changing the regularization parameters on a) the average Kappa value computed over the 58 benchmark problems, and b) the average absolute weight dissimilarity between two consecutive abstract layers. The horizontal axes depict the β and ψ parameters attached to the $\mathcal{R}_1$ and $\mathcal{R}_2$ regularizers, respectively.

Table 2
Parameter settings of all classifiers used in the comparisons.

Classifier	Parameters
LR	Regularization = L_2, solver = lbfgs, C = 1, max_iterations = 100
GNB	No parameter reported
DT	Split criterion = Gini impurity
SVM	Kernel = RBF, $\gamma = 1/number\_features \times number\_cases$
RF	Number of estimators = 100, max_features = $\sqrt{number\_features}$
MLP	Activation function = ReLu, hidden layers = 2, neurons for each hidden layer = 50, 200 epochs
ELTCN	Activation function = $tanh, T$ = 5, $\beta_{ij} = 0.01, \psi_i = 0.01$, 200 epochs

valuable pieces of information that help assess the performance of machine learning algorithms. In our research, we use default parameter values since our ultimate goal is not to outperform the state-of-the-art classifiers but rather focus on the model's interpretability. Therefore, obtaining competitive prediction rates should be seen as an indicator that the explanations that the ELTCN model produces are trustworthy.

The datasets adopted for simulations and the decision of using fixed hyperparameters lead to the first research hypothesis: "the proposed ELTCN model (together with our backpropagation learning method) attains competitive prediction rates concerning traditional classifiers when no hyperparameter tuning is performed". The second research hypothesis is concerned with the interpretability component and reads as follows: "the proposed ELTCN model allows elucidating its decision process through its knowledge structures". It should be mentioned that the second hypothesis will be empirically investigated with the aid of a proof of concept (see Section 7.5).

7.4. Statistical analysis

As the first experiment, we compare the performance of the proposed model when using the sigmoid function (ELTCN-S) and the hyperbolic tangent function (ELTCN-H). Fig. 4 shows the average Kappa values for both variants on the 58 datasets adopted for simulation, after performing 10-fold cross-validation. The results indicate that the ELTCN-H variant is more accurate in terms of Kappa values as it reports a bigger performance area. This result is somehow expected since the sigmoid transfer function often causes the neurons to quickly saturate either toward zero or one, thus limiting their overall prediction

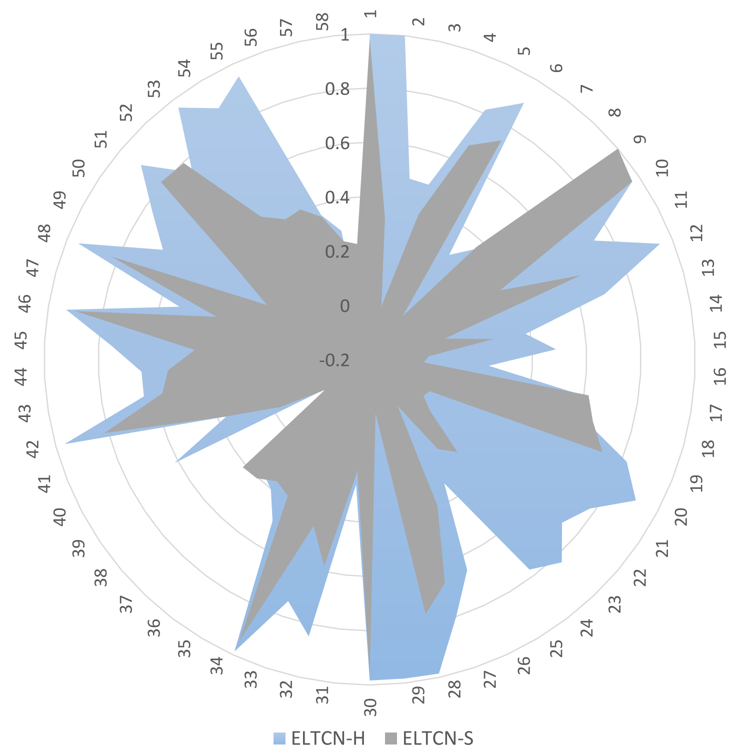

Fig. 4. Average Kappa values reported by our model for both transfer functions on the 58 datasets. The results indicate that the ELTCN-H variant is more accurate in terms of Kappa values as it reports a bigger performance area.

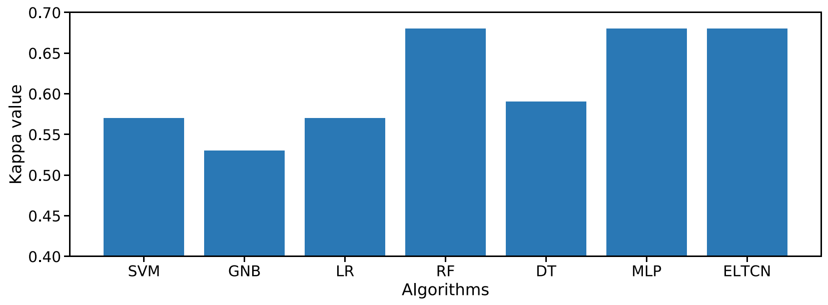

Fig. 5. Average Kappa value achieved by each classifier on the 58 datasets. MLP, RF and ELTCN report the highest prediction rates in this study.

capability. Hereinafter, ELTCN will refer to the proposed model using hyperbolic tangent functions and a *softmax* output layer.

Next, we compare the ELTCN algorithm against the state-of-the-art models aimed at solving traditional pattern classification problems. Fig. 5 depicts the average Kappa values attained by each classification model, after performing 10-fold cross-validation. Observe that MLP, RF and ELTCN report the highest prediction rates (expressed in terms of Kappa values) in this study. This comes with no surprise because both MLP and RF are powerful classifiers. However, they operate as black boxes, which means that they cannot explain by themselves how their decisions were made.

To determine whether the observed performance differences are statistically significant or not, we run the Friedman [40] test. This non-parametric test will reject the null hypothesis when at least two classifiers report significant different performance differences. The p-value = 0.0 < 0.05 advocates for the rejection of the null hypothesis for a 95% confidence interval.

Next, we conduct pairwise comparisons with ELTCN being the control algorithm. Such pairwise comparisons are conducted using the Wilcoxon [41] signed-rank test. Table 3 shows the p-values reported by this non-parametric test, the negative (R^-) and the positive (R^+) ranks, the corrected p-values according to Bonferroni-Holm (as suggested by Demšar [42] and Garcia and Herrera [43]). This method is used for multiple comparisons and controls the *family-wise error rate*. The last column in Table 3 indicates whether the null hypothesis was rejected or not for a significance level of 0.05.

The statistical analysis suggests that our proposal significantly outperforms all white boxes (which are deemed interpretable to some extent) and the SVM classifier while remaining competitive with RF and MLP. This result advocates for the acceptance of the first research hypothesis, namely: "the proposed ELTCN model performs comparably to the selected black boxes when no hyperparameter tuning is conducted". However, being able to outperform the traditional classifiers is not the ultimate goal of our research. Instead, we are interested in exploiting ELTCNs' interpretability.

7.5. Understanding the predictions

In this subsection, we discuss how to interpret the ELTCN-based classifiers and how to produce global explanations. In order to do that, we adopt the well-known Iris dataset (see Table 1) such that all instances will be used to train our neural system. In consequence, the cross-validation procedure will not be performed in the following experiments.

The first step toward building the model concerns the construction of an initial weight matrix and the design of constraints that regulate which weights should be preserved. This process often involves domain experts, thus enabling the human-machine interaction. In this paper, the human knowledge is replaced with Pearson's correlation. Fig. 6 portrays the Pearson's correlation between the problem variables, the correlation coefficients and a mask matrix. The latter matrix indicates which weights should be preserved (to a large extent) during the backpropagation learning process.

The training accuracy (in terms of Kappa) reported by the model is 0.98, which suggests that we can have a reasonable degree of confidence in the knowledge captured by the network. Fig. 7 depicts the normalized weights that regulate the interaction between the input neural concepts across $T = 5$ abstract layers. The symbol * indicates that the weight connects two correlated variables according to the mask matrix, thus we should expect little variability in these weights. The simulation results show that the $\mathcal{R}_1$ regularizer in Eq. (17) effectively enforces this behavior.

Fig. 8 shows the weights and bias components through the different abstract layers. It can be confirmed that the patterns enforced by the $\mathcal{R}_1$ regularizer match with the mask matrix depicted in Fig. 6.

The weights depicted in Fig. 7 provide insight into the dynamic behavior of our neural system. However, after performing T iterations, the whole ELTCN model can be expressed by means of N non-linear regression models. Eqs. (23)–(25) show the raw regression models (i.e., before computing the final values with the transfer function) related to *iris-setosa* (y_1), *iris-virginica* (y_2) and *iris-versicolor* (y_3), respectively,

$$\bar{y}_1 = 0.9a_1^{(T)} - a_2^{(T)} - 0.55a_3^{(T)} + 0.3a_4^{(T)} + h_1^{(T)} \tag{23}$$

$$\bar{y}_2 = -0.1a_1^{(T)} + 0.13a_2^{(T)} - 0.23a_3^{(T)} + 0.6a_4^{(T)} - h_2^{(T)} \tag{24}$$

Table 3
Wilcoxon pairwise analysis with Bonferroni-Holm correction for a significance level of 0.05 where the ELTCN classifier is used as the control method.

Algorithm	*p*-value	R^-	R^+	Bonf-Holm	Hypothesis
SVM	8.627E−06	13	42	2.588E−05	Reject
GNB	1.240E−07	11	46	7.438E−07	Reject
LR	2.203E−06	13	42	1.101E−05	Reject
RF	6.449E−01	28	27	6.875E−01	Fail to reject
DT	5.120E−06	8	48	2.048E−05	Reject
MLP	3.437E−01	33	22	6.875E−01	Fail to reject

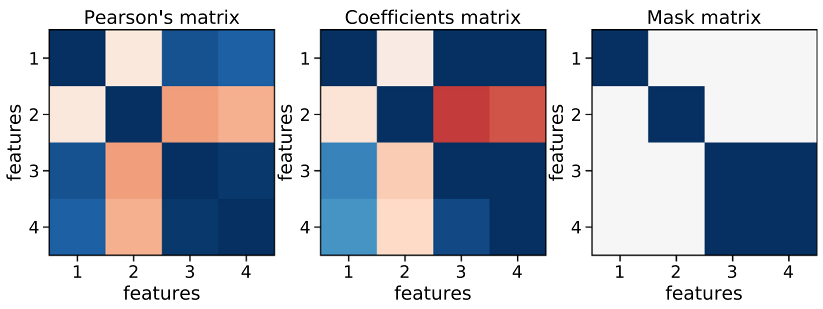

Fig. 6. Pearson's correlation between variables in the Iris dataset (the color scale moves from red to blue to indicate the transition from negative to positive values, zero is light gray). Indexes in vertical and horizontal axes denote variables (x_1 and x_2 are the length and width of the sepal, while x_3 and x_4 are the length and width of the petal, respectively). The mask matrix indicates where the absolute value of correlation between the variables is higher than a given threshold (equal to 0.9 in this example).

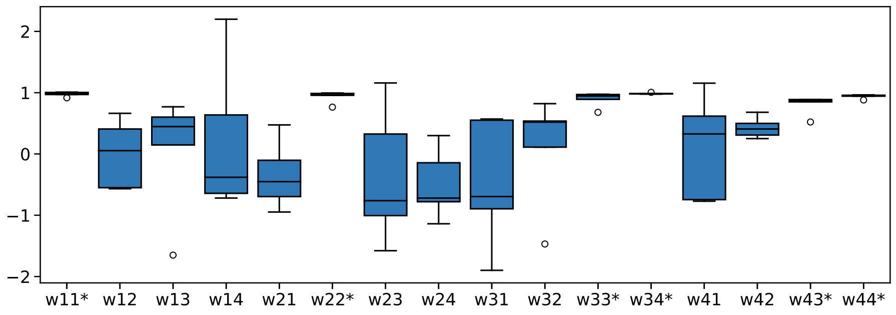

Fig. 7. Behavior of weights in the ELTCN model for the Iris datasets across the abstract layers. The symbol * means that the weight connects two correlated variables, thus it is not desirable for the weight to change much during two consecutive iterations.

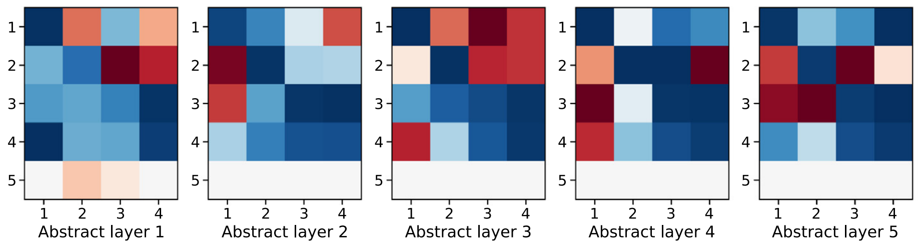

Fig. 8. Weight and bias values across the different abstract layers in the ELTCN model. The color scale moves from red to blue to indicate transition from negative to positive values. Indexes in vertical and horizontal axes denote variables (x_1 and x_2 are the length and width of the sepal, while x_3 and x_4 are the length and width of the petal, respectively). The last index of each vertical axis denotes the bias component.

$$\bar{y}_3 = 0.003a_1^{(T)} + 0.2a_2^{(T)} + 0.4a_3^{(T)} - 0.75a_4^{(T)} + h_3^{(T)} \tag{25}$$

where $h_1^{(T)} = 3.59\text{E}-5, h_2^{(T)} = 5.49\text{E}-6$ and $h_3^{(T)} = 1.87\text{E}-5$. The reader can observe that the offset values in these equations are quite small as a result of the $\mathcal{R}_2$ regularizer in Eq. (17). This implies that the predictions can be primarily explained by the weights, which have been normalized as explained in Section 6 to facilitate their interpretation.

Eqs. (23)–(25) allow characterizing the ELTCNs's after having captured the dynamics of the problem. However, we can go one step further and explain why the model preferred one decision class over another. Let us suppose that our model determined that *iris-versicolor* was the most likely decision class for a given instance. We can derive explanations to understand why the ELTCN model did not classify that instance as *iris-virginica*. This can be done by looking at the dissimilarity between the weights attached to the same neural concepts in Eqs. (24) and (25). A closer inspection of these equations reveals that weights associated with the third and fourth neural concepts play a pivotal role when determining the decision class for a new instance.

Eq. (26) shows a new measure that allows determining the relevance of each neural concept in the classification using the dissimilarity of the weights connected to the decision classes as a proxy,

$$\sigma_k(y_i, y_j) = \frac{|w_{k_{(y_i)}} - w_{k_{(y_j)}}|}{\sum_{l=0}^{M} |w_{l_{(y_i)}} - w_{l_{(y_j)}}|} \tag{26}$$

where $0 \leqslant \sigma_k \leqslant 1$ is the relevance of the k-th neural concept and $w_{k_{(y_i)}}$ denotes the value of the k-th weight in the raw regression model associated with the i-th decision class. The rationale of this interpretability measure is that, if two weights connected to the same input neuron c_k are reasonably similar, then c_k should not be responsible for the different outcomes.

Fig. 9 portrays the relevance of the neural concepts attached to the Iris dataset. The results confirm that the third and fourth neurons (the ones denoting the length and width of the petal, respectively) are pivotal when deciding between *iris-versicolor* and *iris-virginica*.

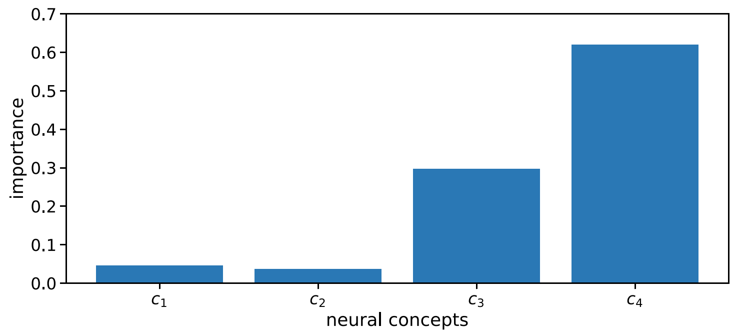

Fig. 9. Relevance of the neural concepts in the Iris dataset problem. This interpretability measure uses the dissimilarity of the output weights as a proxy.

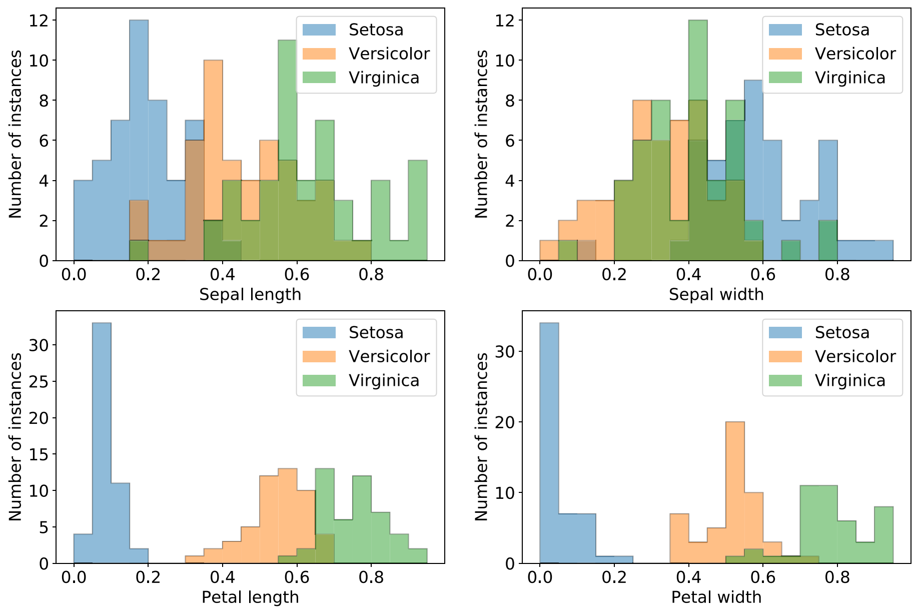

Fig. 10. Distribution of instances of the Iris dataset according to the problem features.

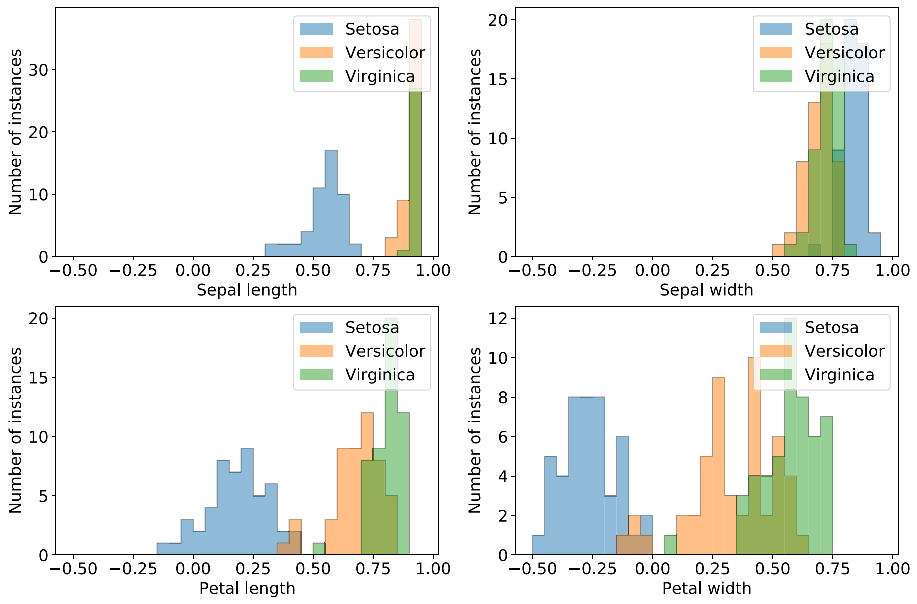

Fig. 11. Distribution of instances of the Iris dataset according to the neural concepts.

It seems opportune to make a distinction between the problem variables and the neural concepts *mapping* these variables. The former ones are static entities while the latter ones are high-level features that capture the dynamics of the system. Fig. 10 shows the distribution of instances according to the problem features, while Fig. 11 shows the distribution according to the high-level features after performing $T = 5$ iterations.

All in all, the proper way of interpreting the ELTCN model is as follows: "*after letting the neural concepts interact with each other T times, we have found that the concepts denoting the petal length and petal width are very important in classifying the instance as iris-versicolor instead of iris-virginica*". Being able to naturally provide such explanations is deemed the most attractive feature of our model. On top of that, the performance of our model proved to be competitive when compared with traditional classifiers.

Before concluding our paper, it seems relevant to discuss the meaning behind the term "interact". When modeling problems where the variables are independent of each other, this term might be confusing since the variables might not interact in a physical sense. However, it does not change the fact that we can use some of these variables to approximate the values of others during the reasoning process. Therefore, the term "interact" refers to the predictive power of the variables describing the problem. In a perfect scenario, the problem should involve dynamic features, the relationships defined by experts should involve an authentic causal meaning and the iterations should have a clear interpretation for the system being modeled. Bringing these features together poses interesting theoretical challenges touching upon increasingly neglected research areas such as knowledge representation and knowledge engineering. Such challenges will be addressed by the authors in future research efforts.

8. Conclusion

In this paper, we have presented a neural model called *Evolving Long-term Cognitive Network* for pattern classification. This network is trained with a backpropagation algorithm that includes two regularization components aimed at increasing the model's interpretability. The first regularization component seeks to reduce the variability of weights from one iteration to another, while the second one is devoted to reducing the impact of the offset parameter on the network's predictions. Besides, we proposed a simple normalization strategy to confine the weights to a certain interval.

The numerical simulations using 58 classification problems have shown that our model performs comparably to MLP and RF while outperforming the other classifiers selected for comparison. These results are in line with our first research hypothesis and confirm that our model is a competitive player in terms of Kappa values, when compared with traditional classifiers and tested with traditional datasets. However, what is truly remarkable is the ability of ELTCNs to explain why one decision was preferred over another. This is done by measuring the dissimilarity between the weights that connect the same neural concepts (high-level features) with different decision classes. In our opinion, the most interesting feature of the explanations is that they are global since they are based on the knowledge representations learned by the network. When putting it all together, we end up having a very flexible classifier that can naturally explain its decision process while also reporting competitive prediction rates.

Of course, our research is far from being complete. The issues related to network construction and the injection of domain knowledge into the model are important challenges to be investigated. Moreover, producing higher prediction rates is always desirable in any classifier. Last but not least, it would be interesting to investigate analytically the trade-off between prediction, stability, and interpretability in the proposed neural system.

CRediT authorship contribution statement

Gonzalo Nápoles: Conceptualization, Methodology, Writing - original draft, Supervision. **Agnieszka Jastrzębska:** Writing - review & editing, Investigation, Supervision. **Yamisleydi Salgueiro:** Data curation, Software, Validation, Visualization.

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.

Acknowledgements

The authors would like to thank the reviewers who provided constructive feedback. Part of this research was supported by the Special Research Fund (BOF) of Hasselt University, Belgium, through the project BOF20KV01. Part of the contribution was supported by the National Science Centre, grant No. 2019/35/D/HS4/01594, decision no. DEC-2019/35/D/HS4/01594.

References

[1] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., 2012..
[2] W. Huang, S. Oh, W. Pedrycz, Fuzzy wavelet polynomial neural networks: analysis and design, IEEE Trans. Fuzzy Syst. 25 (5) (2017) 1329–1341.
[3] J.A. Vargas, W. Pedrycz, E.M. Hemerly, Improved learning algorithm for two-layer neural networks for identification of nonlinear systems, Neurocomputing 329 (2019) 86–96.
[4] E. López-Rubio, F. Ortega-Zamorano, E. Domínguez, J. Muñoz-Pérez, Piecewise polynomial activation functions for feedforward neural networks, Neural Process. Lett. 50 (1) (2019) 121–147.
[5] P. Maji, R. Mullins, On the reduction of computational complexity of deep convolutional neural networks, Entropy 20(4)..
[6] B.A. Marquez, L. Larger, M. Jacquot, Y.K. Chembo, D. Brunner, Dynamical complexity and computation in recurrent neural networks beyond their fixed point, Scientific Rep. 8 (1) (2018) 3319.
[7] N.F. Hardy, D.V. Buonomano, Encoding time in feedforward trajectories of a recurrent neural network model, Neural Comput. 30 (2) (2018) 378–396.
[8] J. Wang, L. Zhang, Q. Guo, Z. Yi, Recurrent neural networks with auxiliary memory units, IEEE Trans. Neural Networks Learn. Syst. 29 (5) (2018) 1652–1661.
[9] P. Liu, Z. Zeng, J. Wang, Multistability of recurrent neural networks with nonmonotonic activation functions and mixed time delays, IEEE Trans. Syst. Man Cybern.: Syst. 46 (4) (2016) 512–523.
[10] H.-G. Han, S. Zhang, J.-F. Qiao, An adaptive growing and pruning algorithm for designing recurrent neural network, Neurocomputing 242 (2017) 51–62.
[11] B. Kosko, Fuzzy cognitive maps, Int. J. Man Mach. Stud. 24 (1) (1986) 65–75.
[12] W. Froelich, W. Pedrycz, Fuzzy cognitive maps in the modeling of granular time series, Knowl.-Based Syst. 115 (2017) 110–122.
[13] G. Nápoles, F. Vanhoenshoven, R. Falcon, K. Vanhoof, Nonsynaptic error backpropagation in long-term cognitive networks, IEEE Trans. Neural Networks Learn. Syst. 31 (3) (2019) 865–875.
[14] G. Nápoles, F. Vanhoenshoven, K. Vanhoof, Short-term cognitive networks, flexible reasoning and nonsynaptic learning, Neural Networks 115 (2019) 72–81.
[15] G. Nápoles, C. Mosquera, R. Falcon, I. Grau, R. Bello, K. Vanhoof, Fuzzy-rough cognitive networks, Neural Networks 97 (2018) 19–27.
[16] E.I. Papageorgiou, A. Kannappan, Fuzzy cognitive map ensemble learning paradigm to solve classification problems: application to autism identification, Appl. Soft Comput. 12 (12) (2012) 3798–3809.
[17] R. Natarajan, J. Subramanian, E.I. Papageorgiou, Hybrid learning of fuzzy cognitive maps for sugarcane yield classification, Comput. Electron. Agric. 127 (2016) 147–157.
[18] P. Christodoulou, A. Christoforou, A.S. Andreou, Improving the performance of classification models with fuzzy cognitive maps, in: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017, 1–6..
[19] H.J. Song, C.Y. Miao, R. Wuyts, Z.Q. Shen, M. D'Hondt, F. Catthoor, An extension to fuzzy cognitive maps for classification and prediction, IEEE Trans. Fuzzy Syst. 19 (1) (2011) 116–135.
[20] K. Guo, R. Chai, H. Candra, Y. Guo, R. Song, H. Nguyen, S. Su, A hybrid fuzzy cognitive map/support vector machine approach for EEG-Based emotion classification using compressed sensing, Int. J. Fuzzy Syst. 21 (1) (2019) 263–273.
[21] G. Nápoles, M. Leon Espinosa, I. Grau, K. Vanhoof, R. Bello, Fuzzy cognitive maps based models for pattern classification: advances and challenges (2018) 83–98..
[22] G. Papakostas, D. Koulouriotis, A. Polydoros, V. Tourassis, Towards Hebbian learning of Fuzzy Cognitive Maps in pattern classification problems, Expert Syst. Appl. 39 (12) (2012) 10620–10629.
[23] J.L. Salmeron, P.R. Palos-Sanchez, Uncertainty propagation in fuzzy grey cognitive maps with Hebbian-like learning algorithms, IEEE Trans. Cybern. 49 (1) (2019) 211–220.
[24] A. Amirkhani, M.R. Mosavi, F. Mohammadizadeh, S.B. Shokouhi, Classification of intraductal breast lesions based on the fuzzy cognitive map, Arab. J. Sci. Eng. 39 (5) (2014) 3723–3732.
[25] A. Kannappan, E.I. Papageorgiou, A new classification scheme using artificial immune systems learning for fuzzy cognitive mapping, in: 2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2013, 1–8..
[26] K. Bhutani, M. Kumar Gaurav, Fuzzy inference & system fuzzy cognitive maps based classification, in: 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), 2015, pp. 305–309.
[27] G. Pajares, J. Sánchez-Lladó, C. López-Martínez, Fuzzy cognitive maps applied to synthetic aperture radar image classifications, in: J. Blanc-Talon, R. Kleihorst, W. Philips, D. Popescu, P. Scheunders (Eds.), Advanced Concepts for Intelligent Vision Systems, 2011, 103–114..
[28] A. Nair, D. Reckien, M.F.A.M. van Maarseveen, Generalised fuzzy cognitive maps: considering the time dynamics between a cause and an effect, Appl. Soft Comput. 92 (2020) 106309.
[29] G. Nápoles, J.L. Salmeron, K. Vanhoof, Construction and supervised learning of long-term grey cognitive networks, IEEE Trans. Cybern. (2019) 1–10.

[30] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks 6 (6) (1993) 861–867.
[31] F. Scarselli, A. Chung Tsoi, Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results, Neural Networks 11 (1) (1998) 15–37.
[32] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: 3rd International Conference for Learning Representations, San Diego, 2015, 1–15..
[33] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: International Conference on Learning Representations, 2019..
[34] E. Kim, S. Oh, W. Pedrycz, Design of reinforced interval type-2 fuzzy C-means-based fuzzy classifier, IEEE Trans. Fuzzy Syst. 26 (5) (2018) 3054–3068.
[35] S.J. Hanson, L.Y. Pratt, Comparing Biases for Minimal Network Construction with Back-Propagation, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems 1, 1989, 177–185..
[36] G. Nápoles, A. Jastrzębska, C. Mosquera, K. Vanhoof, W. Homenda, Deterministic learning of hybrid Fuzzy Cognitive Maps and network reduction approaches, Neural Networks 124 (2020) 258–268.
[37] J. Alcalá, A. Fernández, 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. Multiple-Valued Logic Soft Comput. 17 (2–3) (2011) 255–287.
[38] M. Lichman, UCI Machine Learning Repository, 2013..
[39] N. Japkowicz, M. Shah, Evaluating Learning Algorithms: A Classification Perspective, 2011..
[40] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Am. Stat. Assoc. 32 (200) (1937) 675–701.
[41] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.
[42] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[43] S. Garcia, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. 9 (Dec) (2008) 2677–2694.