Neural Networks 136 (2021) 40–53

Contents lists available at ScienceDirect

Neural Networks

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

Smooth dendrite morphological neurons

Wilfrido Gómez-Flores [a,*], Humberto Sossa [b,c]

[a] Centro de Investigación y de Estudios Avanzados del IPN, Unidad Tamaulipas, Parque TECNOTAM, ZIP 87130, Ciudad Victoria, Tamaulipas, Mexico
[b] Instituto Politécnico Nacional, CIC, Av. Juan de Dios Bátiz S/N, Col. Nueva Industrial Vallejo, Gustavo A. Madero, ZIP 07738, Mexico City, Mexico
[c] Tecnológico de Monterrey, Campus Guadalajara, Av. Gral. Ramón Corona 2514, ZIP 445138, Zapopan, Jalisco, Mexico

ARTICLE INFO

Article history:
Received 1 June 2020
Received in revised form 18 December 2020
Accepted 21 December 2020
Available online 29 December 2020

Keywords:
Morphological neurons
Dendrite processing
Hyperbox-shaped dendrite
Neural networks
Smooth activation functions

ABSTRACT

A typical feature of hyperbox-based dendrite morphological neurons (DMN) is the generation of sharp and rough decision boundaries that inaccurately track the distribution shape of classes of patterns. This feature is because the minimum and maximum activation functions force the decision boundaries to match the faces of the hyperboxes. To improve the DMN response, we introduce a dendritic model that uses smooth maximum and minimum functions to soften the decision boundaries. The classification performance assessment is conducted on nine synthetic and 28 real-world datasets. Based on the experimental results, we demonstrate that the smooth activation functions improve the generalization capacity of DMN. The proposed approach is competitive with four machine learning techniques, namely, Multilayer Perceptron, Radial Basis Function Network, Support Vector Machine, and Nearest Neighbor algorithm. Besides, the computational complexity of DMN training is lower than MLP and SVM classifiers.

1. Introduction

An artificial neuron is a mathematical function inspired by the biological neurons in the animals' nervous system, which emulates dendrites (inputs), soma (integrating unit), and axons (outputs).

The perceptron is probably the most known artificial neuron model in the literature, which is a linear discriminant. For a d-dimensional input pattern $\mathbf{x} = [x_1, \ldots, x_d]^T$, the perceptron's input–output mapping is

$$g(\mathbf{x}) = \varphi\left(\mathbf{w}^T\mathbf{x} + w_0\right), \tag{1}$$

where $\mathbf{w} = [w_1, \ldots, w_d]^T$ is the vector of synaptic weights, w_0 is the bias, and $\varphi(\cdot)$ is an activation function that restricts the response of the neuron. The perceptron's weights are usually determined by minimizing a loss function using a gradient descent scheme (Duda et al., 2012).

It is well-known that a single perceptron is incapable of solving nonlinearly separable problems such as the XOR gate. However, layered perceptrons enable nonlinear responses, which raises the Multilayer Perceptron (MLP) model. In this way, the XOR problem is solved by an MLP with two perceptrons in the hidden layer and one perceptron in the output layer, using the Heaviside step function as an activation function (Minsky & Papert, 1969).

To improve neural computing models, the use of peripheral structures of neurons, such as dendrites, have been investigated as processing units and not solely for data delivery. In this sense, Ritter and Urcid (2003) proposed a dendritic structure called Single Layer Morphological Perceptron (SLMP) for binary classification problems. The authors stated that an SLMP is capable of representing any compact set (closed and limited) in $\mathbb{R}^d$ with accuracy $\epsilon > 0$. Following such a statement, the XOR problem is solved with one SLMP containing two dendrites, and no hidden layers are necessary. In general, any arbitrary two-class problem can be solved with an SLMP. Later, this dendritic model was extended to multiclass problems to classify more than two compact regions (Ritter & Schmalz, 2006).

From a geometrical perspective, a dendrite can be seen as an isooriented hyperbox in $\mathbb{R}^d$ that encloses a subset of input patterns. Besides, each dendrite is labeled with the actual class of enclosed patterns. Hence, a dendrite morphological neuron (DMN) creates decision boundaries based on hyperboxes instead of hyperplanes. DMN training is relatively easy since it consists of finding the extreme vertices of hyperboxes (Sossa & Guevara, 2014). Therefore, the DMN architecture could be simpler than the MLP model and is also capable of obtaining nonlinear decision boundaries.

The morphological notion in DMN is taken from the mathematical morphology since minimum and maximum operations are involved. These operations determine whether an input pattern falls within the region bounded by a dendrite. Then, such a pattern is classified into the class of the dendrite with the highest activation response.

* Corresponding author.
E-mail address: wgomez@cinvestav.mx (W. Gómez-Flores).

https://doi.org/10.1016/j.neunet.2020.12.021
0893-6080/

The maximum and minimum are hard functions that inherently produce decision boundaries with sharp edges comparable to piecewise linear functions. This response is because the maximum and minimum functions force the decision boundary to match the hyperboxes' faces. Besides, depending on the training algorithm, the boundary sharpness risks overfitting because it learns the particularities of the training patterns, which becomes more critical when class distributions overlap.

In this paper, we propose to replace the hard maximum and minimum functions by smooth versions aiming to improve the generalization of the DMN. A hyperbox-based DMN model trained with a divide and conquer strategy is used. Besides, the DMN hyperparameters are tuned through the Differential Evolution algorithm.

The organization of the paper is as follows. Section 2 presents a review of morphological neural networks and hybrid models. Section 3 gives the basic concepts of dendrite morphological neurons, describes a training algorithm, and presents an analysis of the decision boundaries produced by these dendritic structures. The proposed dendrite morphological neuron activated by smooth maximum and minimum functions is introduced in Section 4. The experimental setup to evaluate the proposed approach is described in Section 5. The experimental results are shown in Section 6. Finally, Section 7 gives concluding remarks and future research directions.

2. Related work

Davidson and Hummer (1993) introduced, by the first time, the Morphology Neural Network (MNN) for template learning in dilation image processing. Afterward, Ritter and Sussner (1996) generalized the MNN model through Lattice theory to obtain the first Single Layer Morphological Perceptron (SLMP) model. This approach solves nonlinearly separable binary classifications problems using hyperplanes parallel to Cartesian axes. Besides, Sussner (1998) extended the SLMP to a multilayer approach and presented a supervised learning algorithm.

In 2003, Ritter and Urcid (2003) proposed the first morphological neuron with dendritic processing, where an isooriented hyperbox (i.e., parallel to Cartesian axes of the input space) defines a dendrite. This neuron is also called SLMP, although its architecture differs from its predecessor (Ritter & Sussner, 1996). The dendritic SLMP model only solves binary classification problems and it was extended to multiclass problems later (Ritter & Schmalz, 2006).

The DMN formulation of Ritter and Urcid (2003), based on Lattice algebra, continues practically unchanged nowadays. However, distinct research directions have been explored for developing efficient methods for DMN training. In this sense, Sussner and Esmi (2011) proposed the Morphological Perceptron with Competitive Neurons for multiclass problems, which incorporates a winner-take-all strategy, given by the argmax function. In this approach, patterns of the same class are enclosed by a hyperbox. If hyperboxes of distinct class overlap, they are broken down into two smaller hyperboxes, until no overlapping hyperboxes with different class labels remain.

Similarly, Sossa and Guevara (2014) presented a DMN training algorithm based on a divide and conquer strategy. This method begins enclosing all patterns in one hyperbox and then divides the hyperbox into 2^d sub-hyperboxes if there is misclassification. After generating all the hyperboxes, if two or more hyperboxes of the same class share a common side, they are merged in a single region.

DMN training has also been addressed from an optimization perspective. Zamora and Sossa (2017) used a Stochastic Gradient Descent (SGD) scheme to refine the dendrite parameters initialized by another heuristic algorithm, such as divide and conquer. Because the morphological operations are non-differentiable, a softmax layer is included at the DMN output to estimate class probabilities. Thus, the cross-entropy between these probabilities and the targets is used as a loss function to be minimized by SGD.

Arce et al. (2018) proposed an evolutionary DMN training based on the Differential Evolution algorithm. A population of DMNs is initialized by the k-means++ algorithm. These initial solutions represent the extreme vertices of hyperboxes that are codified into real-valued vectors. Next, the genetic operators of mutation, crossover, and selection improve the population fitness, where the error rate is minimized.

Morphological neurons have also been applied to the regression task. In the work of Araujo (2012), a neural model called Increasing Morphological Perceptron (IMP) is proposed for predicting forecasting stock markets, in which a linear activation function replaces the argmax function. A gradient descent scheme is used to train the IMP neuron.

In another research direction, other kinds of geometric structures to define dendrites have been explored. Barmpoutis and Ritter (2007) proposed the Orthonormal Basis Lattice Neural Network (OB-LNN) to obtain rotated hyperboxes that store information about the classes' local orientation. OB-LNN training consists of finding the best dendrite parameters and rotation matrices. With this approach, it is possible to obtain a reduced number of hyperboxes.

Ritter et al. (2014) introduced the Single-Layer Lattice Perceptron (SLLP) model, where dendrites use L_1 and L_∞ norms as a replacement of the min/max operations. This modification generates polytopes as decision boundaries. Elimination and merge procedures are current techniques for SLLP training.

In the Dendrite Ellipsoid Neuron (DEN) model, Arce et al. (2017) replaced hyperboxes by hyperellipses aiming to obtain smooth decision boundaries. DEN performs dendritic processing but not morphological operations since the Mahalanobis distance substitutes the original DMN formulation based on Lattice algebra. An input pattern is classified into the dendrite class with the minimum Mahalanobis distance. Thus, the decision boundaries produced by DEN are similar to those obtained by Gaussian mixture models (Arce et al., 2019)

Hybrid models that combine DMN with classical perceptrons have been developed. Pessoa and Maragos (2000) proposed the Morphological/rank/linear Neural Networks (MRL-NNs) that employ classical perceptrons. The output of each layer is a convex combination of linear and rank operations applied to inputs. This approach was applied to handwritten character recognition.

The Dilation–Erosion-Linear Perceptron (DELP) is a hybrid neural model proposed by Araujo et al. (2013), which combines a nonlinear operator (dilation and erosion operators) and a linear operator (finite impulse response). Also, the hyperparameters of the model are tuned by the Particle Swarm Optimization algorithm. DELP is limited to binary classification problems.

Recently, Hernández et al. (2020) introduced two hybrid neural models called Morphological-Linear Neural Network (MLNN) and Linear-Morphological Neural Network (LMNN). The former consists of a hidden layer of morphological neurons and an output layer of classical perceptrons. The latter is composed of hidden layers of perceptrons, and morphological neurons at the output layer. Both approaches are trained with SGD, and they have the capability of extracting features.

The Hybrid Morphological/Linear Perceptron (HMLP), proposed by Sussner and Campiotti (2020), is a two-layer model whose hidden layer includes morphological units and classical semi-linear neurons, and linear nodes in the output layer. HMLP is trained via an extreme learning machine approach. This neural model is capable of representing non-differentiable functions.

In another recent work, Mondal et al. (2019) proposed a hybrid neural architecture called Dense Morphological Network

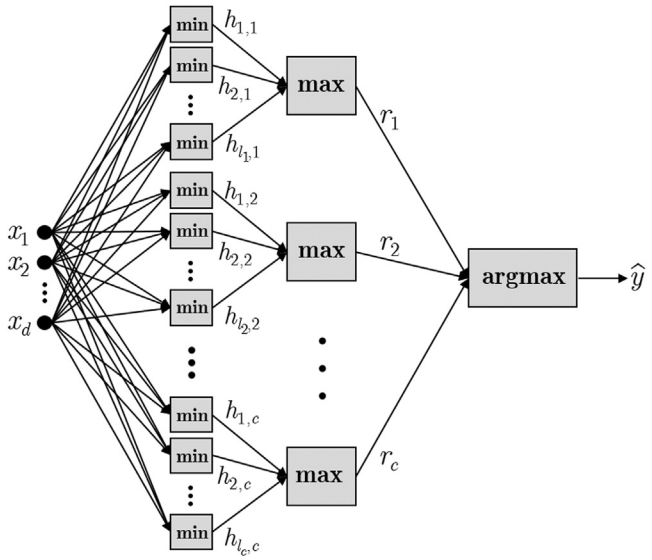

Fig. 1. Topology of a dendrite morphological neuron. The activation of the jth dendrite in the kth class is $h_{j,k}$, for $j = 1, 2, \ldots, l_k$ dendrites and $k = 1, 2, \ldots, c$ classes. The dendrite cluster response for the kth class is r_k. The neuron's output is an argmax function to predict the class label $\hat{y} \in \Omega$ of the input pattern $\mathbf{x} = (x_1, \ldots, x_d)^T$.

(DenMo-Net), consisting of a single layer of morphological neurons followed by a linear combination layer that can approximate smooth functions. The authors used softened morphological functions to make them differentiable for enabling gradient descent training.

Different variants of morphological neurons have been proposed to improve generalization in multiclass and nonlinearly separable problems. Furthermore, dendritic processing models have shown competitive results concerning MLP-based models and other machine learning methods. Note that the current trend is using hybrid neural models to attempt approximating the fundamental structure of the data. Nonetheless, the complexity of hybrid DMN-based neural networks increases and, consequently, training these models also becomes more complex. Therefore, in this paper, the typical hyperbox-based morphological neuron is maintained. However, smooth maximum and minimum functions are used to improve the generalization capacity of the DMN model. The goal is to reduce the sharp decision boundaries obtained with hard maximum and minimum functions.

3. Basics of DMN

3.1. DMN formulation

Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a training set with n observations, where the ith sample is a d-dimensional vector $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})^T$, which is associated to a class label $y_i \in \Omega = \{\omega_1, \ldots, \omega_c\}$, where c is the total number of classes, such that $\mathbf{y} = \{y_1, \ldots, y_n\}$ is the set of training labels.

A DMN is a single-layer dendritic structure, where isooriented hyperboxes represent dendrites in an $\mathbb{R}^d$ space. Each dendrite is labeled with the class of its enclosed training patterns. The most active dendrite is the one that contains (or is closest to) an input pattern $\mathbf{x}$. Then, the predicted class label $\hat{y} \in \Omega$ is determined by the argmax function. The Fig. 1 shows the typical topology of a DMN.

The DMN output is the predicted class to the input pattern $\mathbf{x}$, which is expressed as

$$\hat{y}(\mathbf{x}) = \underset{k=1,\ldots,c}{\arg\max} \left(r_k(\mathbf{x})\right), \tag{2}$$

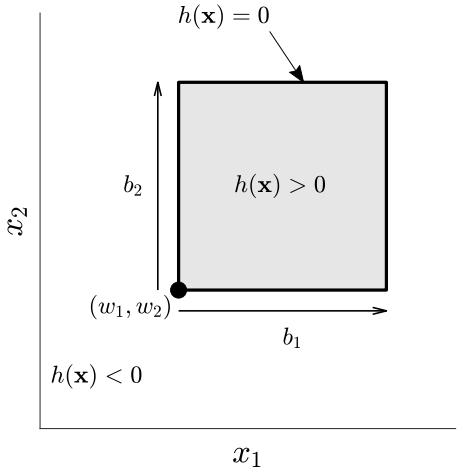

Fig. 2. An isooriented hyperbox in $\mathbb{R}^2$ defined by its dendrite parameters $\mathbf{w} = (w_1, w_2)^T$ and $\mathbf{b} = (b_1, b_2)^T$. The function $h(\cdot)$ activates the dendrite by using the minimum function. The dendrite response is positive when the input pattern $\mathbf{x}$ is inside of the hyperbox, it is zero when $\mathbf{x}$ is on the hyperbox boundary, and it is negative when $\mathbf{x}$ is outside of the hyperbox.

where r_k is the dendrite cluster response for the kth class:

$$r_k(\mathbf{x}) = \max_{j=1,\ldots,l_k} \left(h_{j,k}(\mathbf{x})\right), \tag{3}$$

where $h_{j,k}$ is the activation of the jth dendrite in the kth class:

$$h_{j,k}(\mathbf{x}) = \min_{i=1,\ldots,d} \left(\min\left(x_i - w_{i,j,k},\, w_{i,j,k} + b_{i,j,k} - x_i\right)\right), \tag{4}$$

where $\mathbf{w}_{j,k} = (w_{1,j,k}, \ldots, w_{d,j,k})^T$ is the coordinate of the lowest extreme point of the hyperbox, and $\mathbf{b}_{j,k} = (b_{1,j,k}, \ldots, b_{d,j,k})^T$ is a vector containing the side lengths of the hyperbox for each dimension (Zamora & Sossa, 2017). Fig. 2 shows the three possible dendrite responses given in Eq. (4). From the observations in Fig. 2, the following definitions are deduced.

Definition 1. An isooriented hyperbox in $\mathbb{R}^d$ is defined by the Cartesian coordinates of its lowest extreme point $(w_1, \ldots, w_d)$, and its side lengths for each dimension $b_i > 0$, for $i = 1, \ldots, d$.

Definition 2. Given a dendrite, represented by an isooriented hyperbox in $\mathbb{R}^d$, its activation function $h(\mathbf{x})$ divides the input space into two regions: the inner region, where $h(\mathbf{x}) > 0$, and the outer region, where $h(\mathbf{x}) < 0$.

Definition 3. A single dendrite, defined by an isooriented hyperbox in $\mathbb{R}^d$, has $p = 2d$ parameters. By extension, the total number of parameters of a DMN with $n_d \in \mathbb{Z}^+$ dendrites is $p_t = n_d p$.

3.2. DMN training algorithm

DMN training involves finding the parameters $\mathbf{w}$ and $\mathbf{b}$ of each dendrite. For this task, herein, we used an approach called Linear Divide and Conquer Method (LDCM) proposed by Sossa et al. (2018). This training method represents a substantial improvement to the former version described by Sossa and Guevara (2014) to deal with high-dimensional data. This algorithmic approach is based on a binary search to reduce the computational complexity from $\mathcal{O}(2^{nd})$ to $\mathcal{O}(nd \log_2 nd)$, where n is the number of training patterns and d the dimensionality.

LDCM starts enclosing all training patterns by the isooriented minimum bounding hyperbox $\mathcal{H}_0 \in \mathbb{R}^d$. A margin distance $m \geq 0$ is used to increase the hypervolume of $\mathcal{H}_0$ for producing better

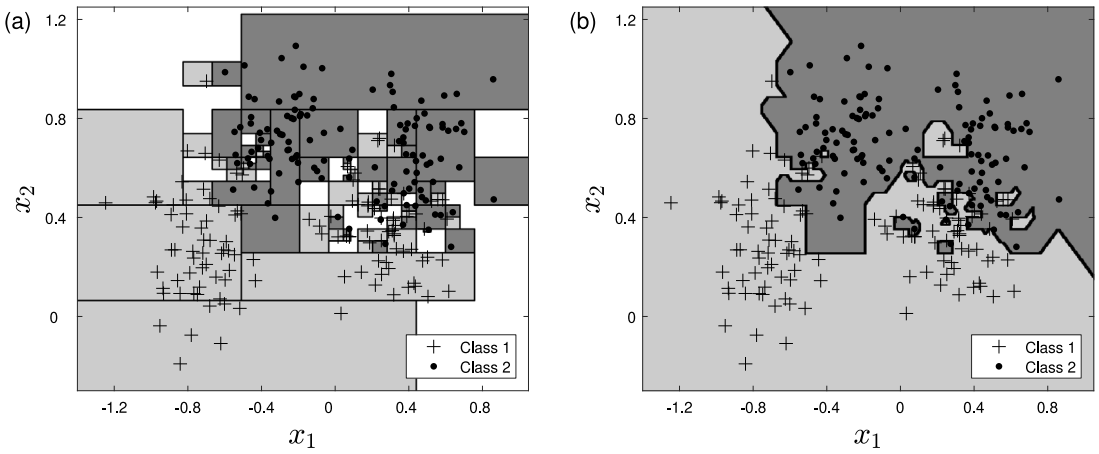

Fig. 3. DMN trained with LDCM on the Ripley dataset ($m = 0.1$ and $\epsilon = 0.01$). (a) Input space divided by hyperboxes. White spaces are ambiguous regions without training patterns. (b) Decision regions determined by the argmax function.

tolerance to noisy test patterns. The margin distance is a multiplicative factor that uniformly extends m times every side of $\mathcal{H}_0$. Hence, the larger the margin distance, the better the tolerance to atypical test patterns.

Next, a recursive process is run to split each dimension of the hyperbox in half and merge adjacent hyperboxes of the same class until the misclassification rate reaches a tolerance error $\epsilon \geq 0$, representing a stop criterion. Hence, the lower the tolerance error, the better the training accuracy. At the final, the hyperboxes coordinates determine the dendrites parameters.

Notice that LDCM has two free hyperparameters (m and ϵ) that should be tuned adequately. For more details of LDCM and numerical examples please refer to Sossa et al. (2018), Sossa and Guevara (2014).

3.3. Decision boundary features

We illustrate some important decision boundary features produced by DMN using the well-known Ripley synthetic dataset (Ripley, 1996).[1] It consists of two classes and two dimensions, where a mixture of two Gaussian distributions generated the data for each class.

Fig. 3 shows the resultant DMN trained with LDCM on the Ripley dataset. The two-dimensional input space is divided into boxes that enclose training patterns of the same class. Also, no boxes are generated in regions without training patterns, so an arbitrary pattern in those ambiguous regions activates its closest dendrite. From dendrite clusters responses, decision regions are determined by the argmax function.

Notably, the decision boundary presents sharp edges like a piecewise linear function. This sharp response is because the maximum and minimum functions, in Eq. (3) and (4), force the decision boundary to match the boxes' sides. In general, when two dendrites clusters respond zero simultaneously for an input pattern $\mathbf{x}$, it implies that $\mathbf{x}$ is on the shared face of two adjacent dendrites of different classes. Hence, $\mathbf{x}$ is on a segment of the decision boundary that is coplanar to the shared faces. From these observations, the following definitions are inferred.

Definition 4. Given a hyperbox-shaped dendrite in $\mathbb{R}^d$, with parameters $\mathbf{w} = (w_1, \ldots, w_d)^T$ and $\mathbf{b} = (b_1, \ldots, b_d)^T$, the dendrite response $h(\mathbf{x})$ is zero if and only if the input pattern $\mathbf{x} = (x_1, \ldots, x_d)^T$ is coplanar to one extreme face at dimension l, that is, $x_l = w_l$ or $x_l = w_l + b_l$, with $1 \leq l \leq d$, and, for the ith dimension, $w_i \leq x_i \leq w_i + b_i$, with $1 \leq i \leq d$ and $i \neq l$.

[1] http://www.stats.ox.ac.uk/pub/PRNN/.

Definition 5. Given two adjacent hyperbox-shaped dendrites in $\mathbb{R}^d$ belonging to different classes, their shared faces simultaneously generate responses equal to zero for an input pattern $\mathbf{x} \in \mathbb{R}^d$ that is on the decision boundary.

On the other hand, DMN learning depends on tuning two hyperparameters of the LDCM algorithm: margin distance m and tolerance error ϵ. Fig. 4 shows the decision boundary variations for distinct values of m and ϵ on the Ripley dataset. Prominently, LDCM is more sensitive to tolerance error than margin distance. If $\epsilon \to 0$, LDCM generates many dendrites, and DMN tends to data overfitting. Contrarily, if $\epsilon \to \infty$, LDCM generates few dendrites, and DMN tends to data underfitting. Thus, it is necessary to incorporate an automatic procedure for tuning the LDCM hyperparameters during DMN training.

In summary, the combination of hard minimum and maximum functions with the characteristics of the training algorithm could generate decision boundaries that roughly represent the natural distribution of patterns. Hence, aiming to improve the generalization capability of DMN, we propose to replace the hard minimum and maximum functions by smooth versions. This modification allows softened decision boundaries.

It is worth mentioning that DMN produces decision surfaces similar to decision trees (DT), i.e., hyperplanes parallel to axes of the input space. The main difference between both approaches is that DMN creates hyperplanes limited to the locality of a hyperbox (dendrite). In contrast, nodes in a DT produce unrestricted hyperplanes that split the entire input space.

4. Proposed approach

4.1. Smooth maximum and minimum functions

A function is considered smooth if it satisfies the following definition (Takacs, 2010).

Definition 6. A smooth function is infinitely many times differentiable; that is, it allows derivatives of any order, and therefore all its derivatives of any order are continuous.

Let $\mathbf{u} = [u_1, \ldots, u_k]$ be an array with k different real numbers and let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function applied to the values in $\mathbf{u}$, such that $f(u_1), \ldots, f(u_k)$. Also, denote the largest value in $\mathbf{u}$ by $u_{\max} = \max(\mathbf{u})$, and the smallest value by $u_{\min} = \min(\mathbf{u})$.

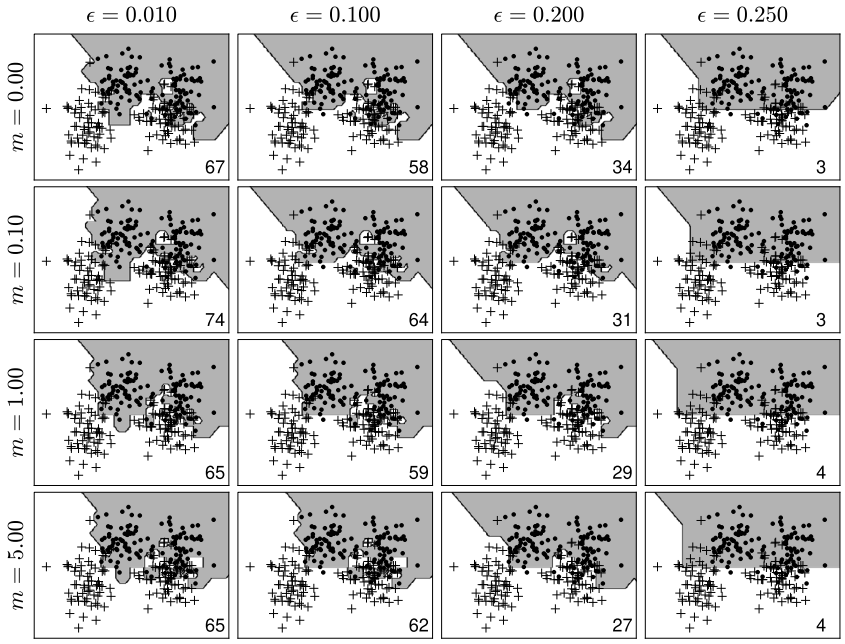

Fig. 4. DMN trained on the Ripley dataset using distinct values of LDCM hyperparameters m and ϵ. The number in the bottom right corner is the total number of dendrites.

If the difference between $u_{\max}$ and the other numbers is large enough, then the following approximation is admissible (Takacs, 2010):

$$\frac{f\left(u_j\right)/f\left(u_{\max}\right)}{\sum_{i=1}^{k} f\left(u_i\right)/f\left(u_{\max}\right)} \approx \begin{cases} 0 & \text{if } u_j \neq u_{\max}, \\ 1 & \text{if } u_j = u_{\max}. \end{cases} \tag{5}$$

If $f(\cdot)$ is smooth and monotonically increasing, a smooth approximation for the maximum function is defined from Eq. (5) as:

$$\max(\mathbf{u}) \approx \sum_{i=1}^{k} \frac{f\left(u_i\right)}{\sum_{j=1}^{k} f\left(u_j\right)} u_i. \tag{6}$$

The exponential function satisfies the characteristics mentioned above, which has the form

$$f(u) = \exp\left(\beta u\right), \quad \beta > 0. \tag{7}$$

Substituting Eq. (7) in (6) gives the smooth maximum function used in this study:

$$\operatorname{smax}(\mathbf{u}) = \sum_{i=1}^{k} \frac{\exp(\beta u_i)}{\sum_{j=1}^{k} \exp(\beta u_j)} u_i, \quad \beta > 0, \tag{8}$$

where the parameter β controls the degree of smoothness. If $\beta \to \infty$, then smax function approximates to max function. Appendix shows that the smax function in Eq. (8) satisfies Definition 6.

Another advantage of the exponential function is that for negative β values, the smooth minimum function is obtained as

$$\operatorname{smin}(\mathbf{u}) = \sum_{i=1}^{k} \frac{\exp(\beta u_i)}{\sum_{j=1}^{k} \exp(\beta u_j)} u_i, \quad \beta < 0. \tag{9}$$

If $\beta \to -\infty$, then smin function approximates to min function.

Fig. 5 shows a comparison of the responses of max and smax with $\beta = 0.5$ and $\beta = 2$. Note that the smooth maximum approximates the hard maximum, although it also rounds off the corners. Besides, the smooth maximum is a convex function, just like the hard maximum. An inverted behavior should be observed for the smooth minimum function.

4.2. Smooth-DMN formulation

The DMN formulation given in Section 3.1 is modified by replacing the minimum and maximum functions by their smoothed versions presented in Section 4.1. Hence, the jth dendrite's smooth activation in the kth class is:

$$\tilde{h}_{j,k}(\mathbf{x}) = \operatorname*{smin}_{i=1,\ldots,d} \left(\operatorname{smin}\left(x_i - w_{i,j,k},\, w_{i,j,k} + b_{i,j,k} - x_i\right)\right), \tag{10}$$

where the dendrite parameters $\mathbf{w}_{j,k}$ and $\mathbf{b}_{j,k}$ are obtained with the LDCM algorithm. Also, the smooth dendrite cluster response for the kth class is:

$$\tilde{r}_k(\mathbf{x}) = \operatorname*{smax}_{j=1,\ldots,l_k} \left(\tilde{h}_{j,k}(\mathbf{x})\right). \tag{11}$$

The decision function in Eq. (2) remains without changes.

The effect of smin and smax functions is to loosen the decision boundary around the faces of the hyperboxes. Thus, this modified version of DMN is called Smooth-DMN (sDMN).

Fig. 6 shows the decision regions generated by sDMN trained with LDCM on the Ripley dataset. Note that the decision boundaries are smooth and track the actual class distributions of training patterns. Furthermore, depending on β value, the decision boundary has different behaviors, providing to sDMN versatility. For $\beta \to 0$, linear decision boundaries are obtained, whereas for $\beta \to \infty$, nonlinear boundaries are generated.

4.3. sDMN tuning algorithm

sDMN training requires finding the dendrite parameters $\mathbf{w}$ and $\mathbf{b}$, and the β value of the smooth minimum and maximum functions. Dendrite parameters are obtained through the LDCM algorithm, which depends on the adjustment of two hyperparameters called margin distance m and tolerance error ϵ. The β hyperparameter only operates in the classification stage and it is adjusted after training the dendrite parameters. Therefore, sDMN learning requires finding the optima m^*, ϵ^*, and β^* hyperparameters from a tuning procedure that evaluates potential solutions.

Note that if the same training set is used to learn $\mathbf{w}$ and $\mathbf{b}$ and to adjust β, the sDMN model could overfit. This is because

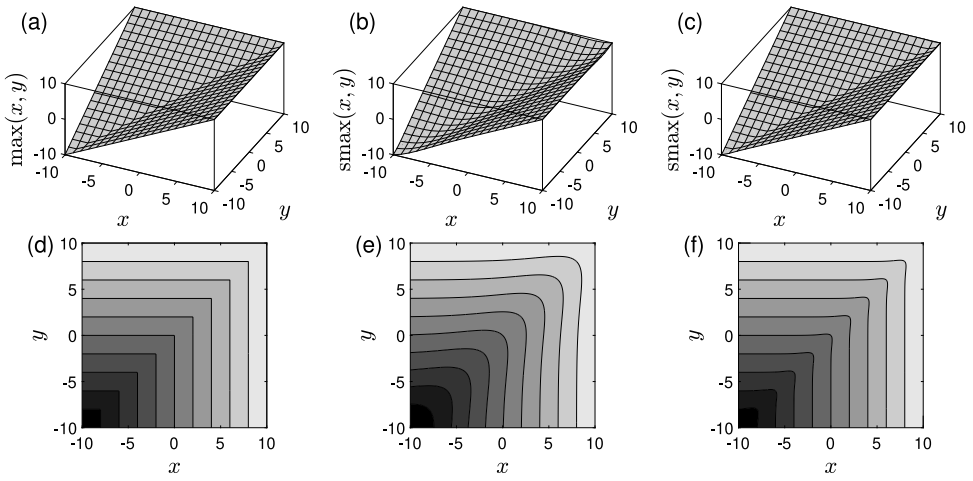

Fig. 5. 3D plots of (a) hard maximum, (b) smax with $\beta = 0.5$, and (c) smax with $\beta = 2$. Contour plots of (d) hard maximum, (e) smax with $\beta = 0.5$, and (f) smax with $\beta = 2$.

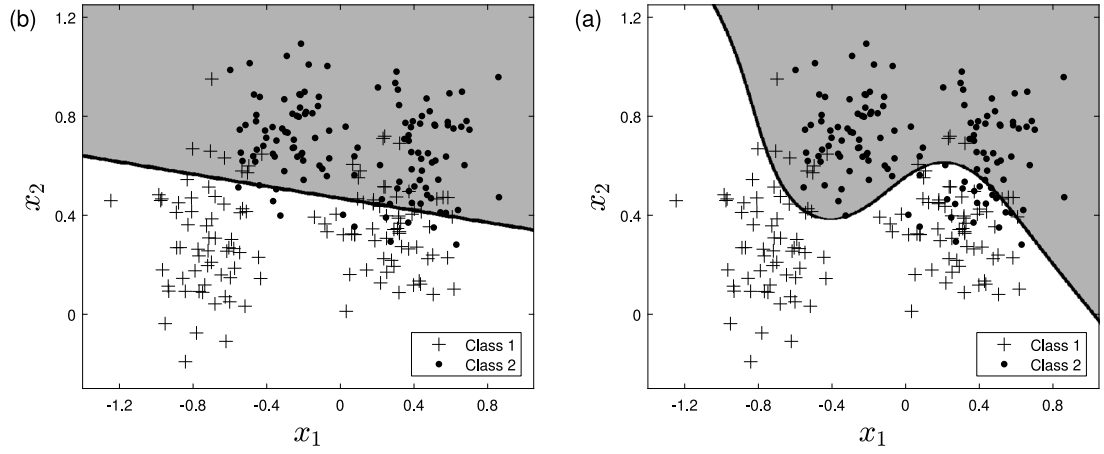

Fig. 6. Ripley dataset classified by sDMN trained with LDCM. (a) Linear decision boundary generated with $\beta = 0.1$ ($m = 0.3$ and $\epsilon = 0.1$). (b) Nonlinear decision boundary generated with $\beta = 2.0$ ($m = 0.1$ and $\epsilon = 0.01$).

β adjustment involves a classification process (given by Eq. (10) and (11)) using $\mathbf{w}$ and $\mathbf{b}$ learned previously on a training set. To avoid overfitting, when searching for the optimal values of m, ϵ, and β, it is convenient using a training subset to learn $\mathbf{w}$ and $\mathbf{b}$ given the pair (m, ϵ), and using another independent training subset to adjust β. Therefore, as shown in Fig. 7, the entire training set $(X, \mathbf{y})$ is randomly divided with stratification into two disjoint subsets $(X_1, \mathbf{y}_1)$, to learn $\mathbf{w}$ and $\mathbf{b}$, and $(X_2, \mathbf{y}_2)$, to adjust β.

The hyperparameters m, ϵ, and β are automatically tuned by the Differential Evolution (DE) algorithm using the DE/rand/1/bin variant (Price et al., 2005). DE is a global optimization algorithm suitable for optimizing real-valued variables. Moreover, DE has been previously used to tune the LDCM hyperparameters by Sossa and Guevara (2014).

The accuracy (success rate) is often maximized during an evolutionary-based training procedure. However, this measure tends to be optimistic due to the high hit rate of the majority class. In the case of unbalanced classes, the Matthews correlation coefficient (MCC) is recommended to measure the classification performance. If classes are equiprobable, the MCC is equal to the accuracy index. Hence, we propose to maximize the MCC instead of the accuracy to deal with the possibility of unbalanced classes. The MCC is expressed as (Gorodkin, 2004)

$$\text{MCC} = \frac{n \cdot \text{tr}(\mathbf{C}) - \sum_{kl} \mathbf{C}_k \mathbf{C}_l}{\sqrt{n^2 - \sum_{kl} \mathbf{C}_k (\mathbf{C}^T)_l} \sqrt{n^2 - \sum_{kl} (\mathbf{C}^T)_k \mathbf{C}_l}}, \tag{12}$$

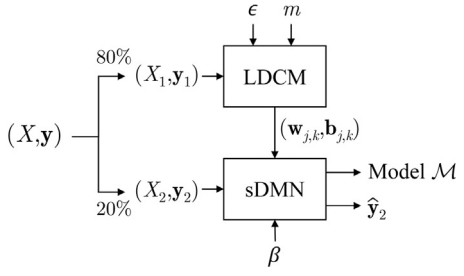

Fig. 7. General pipeline of sDMN training for tuning m, ϵ, and β hyperparameters, where $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is the training set with n samples, and $\mathbf{y} = \{y_1, \ldots, y_n\}$ the corresponding class labels. The model $\mathcal{M}$ comprises the parameters $(\mathbf{w}_{j,k}, \mathbf{b}_{j,k})$, for $j = 1, 2, \ldots, l_k$ dendrites and $k = 1, 2, \ldots, c$ classes, and the β value. $\hat{\mathbf{y}}_2$ denotes the predicted class labels to patterns in X_2 used in model evaluation.

where $\mathbf{C}$ is a multiclass confusion matrix of size c-by-c, n is the number of patterns, $\mathbf{C}_k$ is the kth row of $\mathbf{C}$, $\mathbf{C}_l$ is the lth column of $\mathbf{C}$, and $\mathbf{C}^T$ is transpose of $\mathbf{C}$. This index should tend to unity to indicate an adequate classification performance.

Algorithm 1 presents the pseudocode based on DE/rand/1/bin for sDMN tuning. The population size is defined as ten times the number of variables to optimize, that is, 30 individuals. Also, the algorithm performs 20×10^3 fitness evaluations. At lines 9 and 18,

given a set of hyperparameters m, ϵ, and β, an sDMN model is obtained using the process shown in Algorithm 2. At line 19, the MCC is measured using the predicted class labels of set X_2.

Besides, to improve the convergence properties of DE, Das et al. (2008) proposed varying the mutation factor (F) and the crossover rate (CR) as follows. At each generation, F is randomly generated in the range (0.5, 1). This scheme allows stochastic variations of the difference vector and thus helps retain population diversity as the search progresses. On the other hand, CR is linearly decreased throughout the generations from 1 to 0.5. If $CR = 1$, then all the elements of the parent vector are replaced by the components of the mutant vector. If CR decreases, the offspring vector inherits more elements of the parent vector. Such a variation of CR helps to explore the search space at the beginning and finely adjust the movements of offspring solutions during the later search stages.

Algorithm 1 sDMN training based on DE/rand/1/bin.

Require: training data $(X, \mathbf{y})$; population size n_p; fitness evaluations n_f
Ensure: best individual $\mathbf{z}^* = [m^*, \epsilon^*, \beta^*]$; best sDMN model $\mathcal{M}^*$

1: Total number of generations: $g_{\max} \leftarrow \text{round}(n_f / n_p)$
2: Number of variables to optimize (m, ϵ, and β): $n_v \leftarrow 3$
3: Split $(X, \mathbf{y})$ randomly into two subsets: $(X_1, \mathbf{y}_1)$ and $(X_2, \mathbf{y}_2)$
4: Initialize parents randomly: $\mathbf{z}_1^0, \ldots, \mathbf{z}_{n_p}^0$
5: **for** $g = 0$ **to** $g_{\max} - 1$ **do**
6: Get random mutation factor: $F \leftarrow 0.5 \cdot (1 + \mathcal{U}(0, 1))$
7: Decay crossover rate: $CR \leftarrow 0.5 + 0.5 \cdot \left(\frac{g_{\max} - g}{g_{\max}}\right)$
8: **for** $i = 1$ **to** n_p **do**
9: Get sDMN parent: $\left[\mathcal{M}^{\mathbf{z}_i^g}, \hat{\mathbf{y}}_2^{\mathbf{z}}\right] \leftarrow \mathcal{T}\left(\mathbf{z}_i^g, (X_1, \mathbf{y}_1), (X_2, \mathbf{y}_2)\right)$
10: Select randomly $r_1 \neq r_2 \neq r_3 \neq i$
11: **for** $j = 1$ **to** n_v **do**
12: **if** $\mathcal{U}(0, 1) < CR \vee j = \mathcal{U}_{\text{int}}(1, n_v)$ **then**
13: $u_{i,j}^{g+1} \leftarrow z_{r_1,j}^g + F \cdot (z_{r_2,j}^g - z_{r_3,j}^g)$
14: **else**
15: $u_{i,j}^{g+1} \leftarrow z_{i,j}^g$
16: **end if**
17: **end for**
18: Get sDMN offspring: $\left[\mathcal{M}^{\mathbf{u}_i^{g+1}}, \hat{\mathbf{y}}_2^{\mathbf{u}}\right] \leftarrow \mathcal{T}\left(\mathbf{u}_i^{g+1}, (X_1, \mathbf{y}_1), (X_2, \mathbf{y}_2)\right)$
19: **if** $\text{MCC}\left(\mathbf{y}_2, \hat{\mathbf{y}}_2^{\mathbf{u}}\right) > \text{MCC}\left(\mathbf{y}_2, \hat{\mathbf{y}}_2^{\mathbf{z}}\right)$ **then**
20: Replace parent by offspring: $\mathbf{z}_i^{g+1} \leftarrow \mathbf{u}_i^{g+1}$, $\mathcal{M}^{\mathbf{z}_i^{g+1}} \leftarrow \mathcal{M}^{\mathbf{u}_i^{g+1}}$
21: **else**
22: Keep parent in the population: $\mathbf{z}_i^{g+1} \leftarrow \mathbf{z}_i^g$, $\mathcal{M}^{\mathbf{z}_i^{g+1}} \leftarrow \mathcal{M}^{\mathbf{z}_i^g}$
23: **end if**
24: **end for**
25: **end for**
26: Get best individual $\mathbf{z}^*$ from population with the highest MCC value
27: Get best sDMN model $\mathcal{M}^*$ related to $\mathbf{z}^*$

5. Experimental setup

5.1. Datasets

In this study, nine synthetic and 28 real-world datasets are used for evaluating the classification performance of methods.

Algorithm 2 sDMN training during hyperparameters tuning by DE.

Require: training subsets $(X_1, \mathbf{y}_1)$ and $(X_2, \mathbf{y}_2)$, solution $\mathbf{z} = [m, \epsilon, \beta]$
Ensure: sDMN model $\mathcal{M}$; predicted labels $\hat{\mathbf{y}}_2$

1: Initialize sDMN structure: $\mathcal{M} \leftarrow \emptyset$
2: Learn dendrites parameters: $\left[\mathbf{w}_{j,k}, \mathbf{b}_{j,k}\right] \leftarrow \text{LDCM}\left((X_1, \mathbf{y}_1), m, \epsilon\right)$
3: Get dendrites responses with $(\mathbf{w}_{j,k}, \mathbf{b}_{j,k}, \beta)$: $\tilde{r}_k\left(\tilde{h}_{j,k}(X_2)\right)$ // Eq. (11)
4: Predict class labels: $\hat{\mathbf{y}}_2 \leftarrow \arg\max\left(\tilde{r}_k\right)$ // Eq. (2)
5: Save lowest extreme points: $\mathcal{M}.\mathbf{w}_{j,k} \leftarrow \mathbf{w}_{j,k} = [w_{1,j,k}, \ldots, w_{d,j,k}]$
6: Save side lengths: $\mathcal{M}.\mathbf{b}_{j,k} \leftarrow \mathbf{b}_{j,k} = [b_{1,j,k}, \ldots, b_{d,j,k}]$
7: Save smooth factor: $\mathcal{M}.\beta \leftarrow \beta$

Table 1
2D synthetic datasets with two or three classes c. The number of training patterns is n and the number of testing patterns is n_t. The ID identifies the datasets in the next.

ID	Dataset name	c	n	n_t
S_1	Concentric	3	1455	162
S_2	Horseshoes	2	1350	150
S_3	Moons	2	1350	150
S_4	Ripley Dataset	2	1000	250
S_5	Two-Gaussians	2	960	240
S_6	Three-Gaussians	3	1440	360
S_7	Two-Spirals	2	1350	150
S_8	Three-Spirals	3	1350	150
S_9	XOR Problem	2	1260	140

We used synthetic datasets because they are useful for visualizing the smooth decision boundaries produced by sDMN in linearly and nonlinearly separable data. All the synthetic datasets are two-dimensional with two or three classes, whose characteristics are summarized in Table 1. Also, their class distributions exhibit distinct complexities to illustrate linear and nonlinear separability, as shown in Fig. 8. These datasets were randomly divided with stratification in training (80%) and test (20%) sets.

On the other hand, real-world datasets are taken from different domains of science and engineering, which are often used for evaluating machine learning algorithms. They were obtained from the public UCI Machine Learning Repository (Dua & Graff, 2017). The datasets present a different number of classes $c = \{2, \ldots, 30\}$ and dimensionality $d = \{4, \ldots, 80\}$, and their characteristics are summarized in Table 2. These datasets were randomly divided with stratification in training (90%) and test (10%) sets. For reproducibility purposes, we make available all the datasets used in this study (Gómez-Flores, 2020).

5.2. Classification performance evaluation

Classification performance is measured using test sets in terms of accuracy (ACC) and Matthews correlation coefficient (MCC). Besides, two experiments are conducted to evaluate the performance of the proposed approach.

In the first experiment, the dendritic models DMN and sDMN are compared. The goal is to show the generalization improvement of sDMN by incorporating smooth maximum and minimum functions. In this experiment, the synthetic datasets are used to visualize the decision boundaries generated by both dendritic approaches. Also, the results from real-world datasets are used to compare both methods statistically. In this sense, the

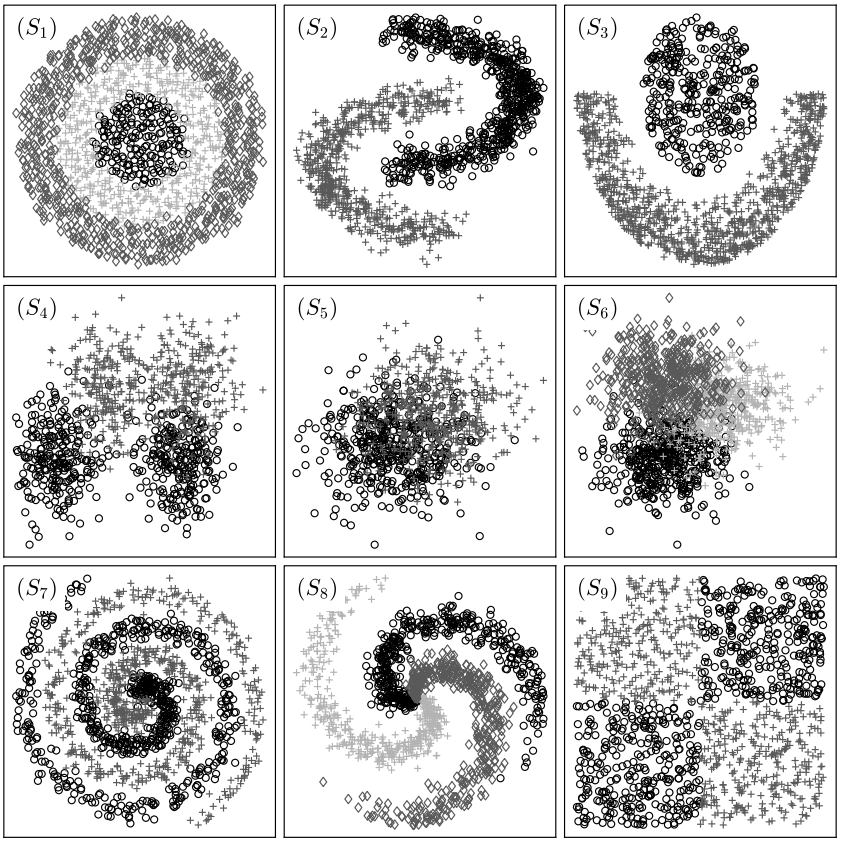

Fig. 8. 2D synthetic datasets with two (+ and ○) or three classes (+ , ○, and ◇): (S_1) Concentric , (S_2) Horseshoes, (S_3) Moons, (S_4) Ripley Dataset, (S_5) Two-Gaussians, (S_6) Three-Gaussians, (S_7) Two-Spirals, (S_8) Three-Spirals, and (S_9) XOR Problem. .

non-parametric Wilcoxon test ($\alpha = 0.05$) is used, whose null hypothesis establishes that data from two continuous distributions have equal medians (Sprent & Smeeton, 2001). It is worth mentioning that the DMN hyperparameters, m and ϵ, are tuned by DE in the same way as in the proposed approach.

On the other hand, the second experiment compares the sDMN classification performance against four traditional machine learning algorithms, namely, Multilayer Perceptron (MLP), Radial Basis Function Network (RBFN), Support Vector Machine with Gaussian kernel (SVM), and k-Nearest Neighbor algorithm (kNN) (Duda et al., 2012). The results from real-world datasets are statistically compared using the Kruskal-Wallis test ($\alpha = 0.05$), which is an extension of the Wilcoxon test to compare more than two groups (Sprent & Smeeton, 2001).

MLP is trained through backpropagation during 1000 epochs. RBFN uses a hybrid learning algorithm in which the k-means algorithm trains the RBF parameters of hidden nodes, and the weights of the output layer are determined by the least-squares pseudoinverse method. SVM is trained with the Sequential Minimal Optimization (SMO) algorithm, which is implemented in the LIBSVM library (Chang & Lin, 2011).

Moreover, for the MLP and RBFN classifiers, it is required to tune the number of hidden neurons. Likewise, SVM requires adjusting two hyperparameters, the penalty constant C and the bandwidth of the Gaussian kernel γ. In kNN, the k-value should be adjusted. Therefore, the adjustment of hyperparameters is achieved using the grid search method (Chang & Lin, 2011), in which the MCC is maximized to find the best values. For MLP and RBFN, the number of hidden neurons are searched in the range $[2, \sqrt{n}]$. For SVM, the ranges are $C = [2^{-5}, 2^{-4}, \ldots, 2^{15}]$ and $\gamma = [2^{-15}, 2^{-14}, \ldots, 2^{3}]$. For kNN, the number of k nearest neighbors is $k = [1, 3, 5, 7, 9]$.

Table 2
Real-world datasets with distinct number of classes $c = \{2, \ldots, 30\}$, and dimensionality $d = \{4, \ldots, 80\}$. The number of training patterns is n and the number of testing patterns is n_t. The ID identifies the datasets in the next.

ID	Dataset name	c	d	n	n_t
R_1	Breast Cancer Wisconsin Diagnosis	2	30	512	57
R_2	Breast Cancer Wisconsin Original	2	9	614	69
R_3	Car Evaluation	4	6	1555	173
R_4	Climate Model Simulation Crashes	2	18	486	54
R_5	Credit Approval	2	15	587	66
R_6	Dermatology	6	34	322	36
R_7	Diabetic Retinopathy Debrecen	2	19	1036	115
R_8	Echocardiogram Data	2	9	55	6
R_9	Glass Identification	6	9	192	22
R_{10}	Heart Disease Cleveland	2	13	267	30
R_{11}	Hepatitis Domain	2	18	101	11
R_{12}	Indian Liver Patient Dataset	2	10	522	57
R_{13}	Ionosphere	2	33	315	36
R_{14}	Iris Data	3	4	135	15
R_{15}	Leaf	30	14	306	34
R_{16}	BUPA Liver Disorders	2	6	306	35
R_{17}	Mice Protein Expression	8	80	497	55
R_{18}	Parkinson's Dataset	2	22	175	20
R_{19}	Pima Indians Diabetes	2	8	691	77
R_{20}	Seeds	3	7	179	20
R_{21}	SPECTF Heart Dataset	2	44	241	26
R_{22}	Statlog (Image Segmentation)	7	18	2079	231
R_{23}	Steel Plates Faults	7	27	1746	195
R_{24}	Thyroid Gland Data	3	5	194	21
R_{25}	Vehicle Silhouettes	4	18	763	83
R_{26}	US Congressional Voting Records	2	16	209	23
R_{27}	Vowel Recognition Data	11	11	891	99
R_{28}	Wine Recognition Data	3	13	160	18

Table 3
Expressions for calculating the number of trainable parameters and computational complexity. n is the number of training patterns, d is the dimensionality, n_d is the total number of dendrites, n_i, n_h, and n_o are the number of input, hidden, and output neurons, respectively, and t is the number of epochs.

Classifier	Trainable parameters	Time complexity
sDMN	$2n_d d$	$\mathcal{O}(nd \log_2 nd)$
MLP	$n_i n_h + n_h n_o + n_h + n_o$	$\mathcal{O}((n_i n_h + n_h n_o)nt)$
RBFN	$n_i n_h + n_h n_o + n_h + n_o$	$\mathcal{O}(n_i n_h n + n_h n_o \min(n_h, n_o))$
SVM	Number of SV times d	$\mathcal{O}(nd^2)$

5.3. Complexity evaluation

The architectural complexity of each classifier is quantified by the number of trainable parameters (NTP), which are the components that shape the decision boundaries generated by a classifier. It is desirable to obtain robust classification models with a reduced NTP (Hernández et al., 2020). In the case of DMN and sDMN, NTP is related to the total number of dendrites. The NTP for MLP is calculated from the total number of synaptic weights and biases in the network. For the RBFN network, NTP considers the centroids and spreads of hidden neurons and the synaptic weights and biases of output neurons. The total number of support vectors (SV) times the dimensionality represents the NTP of an SVM.

In addition to the architectural complexity of classifiers, it is important to measure the training time required to obtain a model. For this task, the CPU wall-clock time is often estimated; however, this measure depends on the characteristics of the computing platform, the programming language, and the developer's programming abilities. An objective measure is the computational complexity, which measures the number of required elementary operations to train a model on n training patterns.

Table 3 summarizes both the NTP and the computational complexity of each classifier's training stage evaluated in this study. It should be considered that kNN does not have an explicit learning phase; instead, training points directly shape decision boundaries. Hence, kNN does not have trainable parameters. Moreover, its computational complexity can only be measured during predicting time, which is $\mathcal{O}(nd + kn)$, where k is the number of nearest neighbors.

The computing platform considered an Intel i9-9900K processor with eight cores at 3.60 GHz, and 64 GB of RAM. All the programs were developed in MATLAB R2020a (The Mathworks, Boston, Massachusetts, USA).

6. Results

The experimental results of the proposed approach are presented in the next. For convenience, discussions are held in this section to interpret the obtained results.

6.1. Experiment 1: DMN vs. sDMN

Table 4 presents the comparison results between DMN and sDMN for the synthetic datasets. Notably, sDMN outperformed DMN in terms of MCC and accuracy (ACC). As expected, the smooth maximum and minimum functions increase the generalization capability of DMN. This effect is observed in Fig. 9, where the decision boundaries generated by sDMN are smooth, while DMN produced coarse and sharp decision boundaries.

The smoothness of the decision boundary is related to the β hyperparameter, which provides to sDMN versatility to generate linear ($\beta < 1$) and nonlinear ($\beta > 1$) boundaries. Thus, the greater the class distribution complexity, the higher the β value. For example, Concentric (S_1) and Two-Spirals (S_8) datasets have very complex shapes, so that $\beta > 8.0$. On the other hand, Three-Gaussians (S_6) dataset has simple class distributions, so that $\beta = 1.25$. Even more, XOR Problem dataset (S_9) is well-separated by two lines, so that $\beta = 0.10$. These results suggest the importance of adequately adjusting the β value according to the class distribution complexity.

Table 4
Experimental results for DMN and sDMN on synthetic datasets. MAD stands for median absolute deviation, and Min and Max stand for minimum and maximum values, respectively. The best results are highlighted in bold.

ID	MCC		ACC		NTP	
	DMN	sDMN	DMN	sDMN	DMN	sDMN
S_1	0.886	**0.927**	0.932	**0.957**	576	**292**
S_2	0.973	**1.000**	0.987	**1.000**	**44**	**44**
S_3	0.982	**1.000**	0.993	**1.000**	64	**48**
S_4	**0.728**	**0.728**	**0.864**	**0.864**	240	**36**
S_5	0.618	**0.675**	0.808	**0.838**	**412**	948
S_6	0.718	**0.722**	0.811	**0.814**	**456**	1188
S_7	0.895	**0.960**	0.947	**0.980**	**364**	460
S_8	0.950	**0.951**	**0.967**	**0.967**	**144**	**144**
S_9	**1.000**	**1.000**	**1.000**	**1.000**	**16**	**16**
Median	0.947	**0.967**	0.895	**0.951**	240	**144**
MAD	**0.064**	0.065	**0.115**	0.118	**173**	342
Max	**1.000**	**1.000**	**1.000**	**1.000**	**576**	1188
Min	0.808	**0.814**	0.618	**0.675**	**16**	**16**

Regarding the number of trainable parameters (NTP), sDMN tended to generate slightly fewer dendrites than DMN. However, for S_5 and S_6 datasets, sDMN generated more than twice the number of dendrites in DMN, which is due to the substantial inter-class overlap that these datasets present. Therefore, sDMN required more dendrites to achieve adequate inter-class separation.

Concerning real-world datasets, Table 5 shows the comparison between DMN and sDMN results, where it is observed that sDMN had better classification performance than DMN. In general, sDMN obtained MCC = 0.820 and ACC = 0.916, while DMN attained MCC = 0.745 and ACC = 0.847. According to the Wilcoxon test ($\alpha = 0.05$), sDMN performed statistically significantly different from DMN. The p-values for MCC and ACC are $p = 0.0336$ and $p = 0.0361$, respectively. This finding confirms that the use of smooth maximum and minimum functions increases the generalization capacity of the proposed approach.

On the other hand, in general, NTP was slightly higher in sDMN than in DMN, although there are no statistically significant differences ($p = 0.9139$) between methods. It should be remarked that the training algorithm for both approaches was LDCM, which is deterministic, but the variation between solutions depends on the hyperparameters found by DE, which is a stochastic search algorithm.

Finally, Fig. 10 shows the behavior of the β hyperparameter for real-world datasets. Most of them required nonlinear decision boundaries ($\beta > 1$), which suggests that class distributions could have complex shapes. This property is verified by Fisher's criterion (J), where the higher the J-value, the better the inter-class separability (Theodoridis & Koutroumbas, 2009). This behavior can also be observed in synthetic datasets, as shown in Fig. 11. For instance, for dataset S_1 (Concentric), the inter-class separability is almost zero and $\beta = 8.71$, which indicates that data is nonlinearly separable and requires a complex decision boundary to separate the classes. Conversely, for dataset S_4 (Two-Gaussians), although the classes present overlap, they can be separated with a simple curve; hence, $\beta = 2.30$. In the case of dataset S_9 (XOR Problem), the J-value is almost zero and β is also almost zero, which indicates that despite the classes are nonlinearly separable, they are separated using two straight lines, shown in Fig. 9. In general, from Fig. 10 and 11, the negative trend suggests that the higher the β value, the lower the inter-class separability.

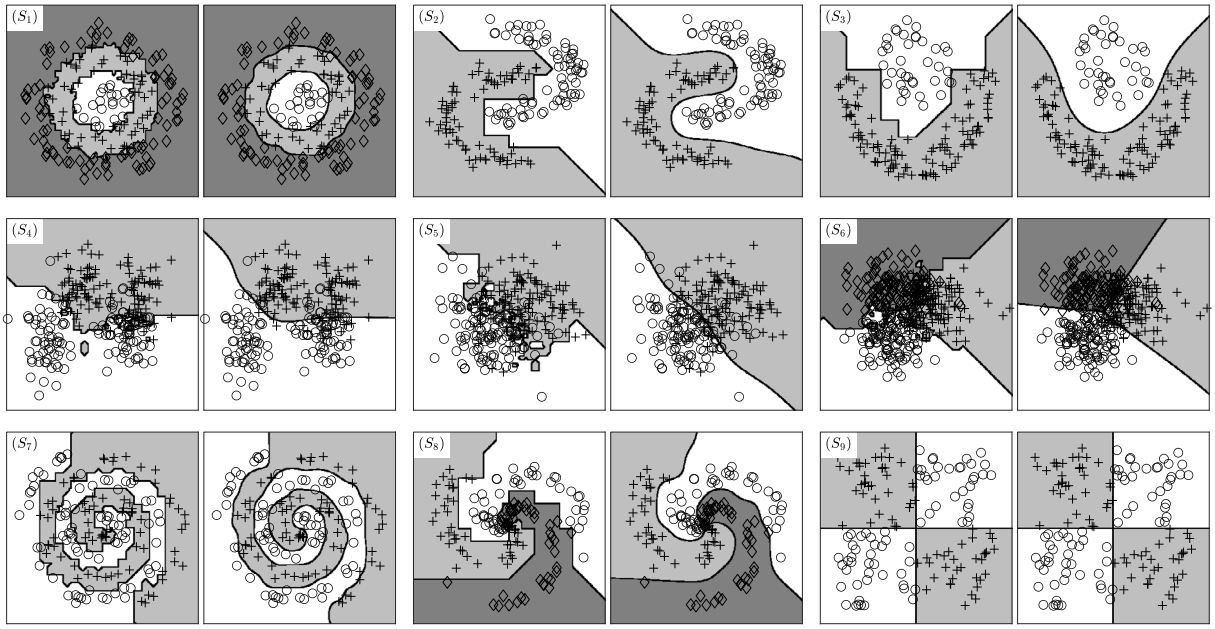

Fig. 9. Decision boundaries generated by DMN (left) and sDMN (right), which are presented in pairs for each synthetic dataset: (S_1) Concentric, $\beta = 8.71$; (S_2) Horseshoes, $\beta = 2.76$; (S_3) Moons, $\beta = 1.82$; (S_4) Ripley Dataset, $\beta = 3.92$; (S_5) Two-Gaussians, $\beta = 2.30$; (S_6) Three-Gaussians, $\beta = 1.22$; (S_7) Two-Spirals, $\beta = 8.54$; (S_8) Three-Spirals, $\beta = 5.62$; and (S_9) XOR Problem, $\beta = 0.10$;.

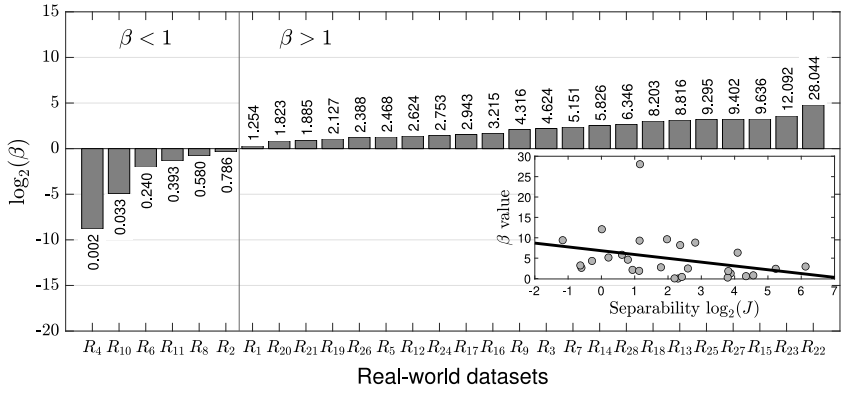

Fig. 10. The behavior of β hyperparameter for real-world datasets. The bars are ordered from lowest to highest β value. The β value without logarithmic compression is shown at the top or bottom of the bars. The subplot shows the linear trend of β value as a function of inter-class separability given by Fisher's criterion.

6.2. Experiment 2: sDMN vs. machine learning algorithms

Table 6 shows the comparison of classification results between sDMN and four machine learning algorithms. In general, SVM had the best classification performance, followed by kNN, MLP, sDMN, and RBFN. However, in terms of MCC, it is notable that sDMN exhibited better consistency (i.e., low median absolute deviation) than its counterparts and obtained the best lower limit, even though all the methods were tuned to maximize MCC. This result indicates that the other classification methods could perform worse than sDMN for some datasets. In the extreme case, for dataset R_{21}, sDMN reached MCC = 0.804, while SVM obtained MCC = 0.129, and MLP achieved MCC = 0.010. This behavior can be due to the methods used for optimizing the classifiers' hyperparameters. While sDMN used DE that is unrestrained to explore the search space, its counterparts used grid search that is constrained to the grid's limits and the granularity for each variable to optimize. Hence, the grid search method may underexplored the search space producing low consistency.

Despite the particular differences between methods, the Kruskal-Wallis test ($\alpha = 0.05$) revealed that, in general, the five classification methods do not present significant statistical differences. The p-values for MCC and ACC are $p = 0.4541$ and

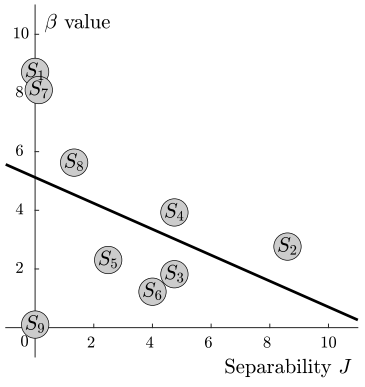

Fig. 11. The behavior of β hyperparameter for synthetic datasets. It is shown the linear trend of β value as a function of inter-class separability given by Fisher's criterion.

Table 5
Experimental results for DMN and sDMN on real-world datasets. MAD stands for median absolute deviation, and Min and Max stand for minimum and maximum values, respectively. The best results are highlighted in bold.

ID	MCC		ACC		NTP	
	DMN	sDMN	DMN	sDMN	DMN	sDMN
R_1	0.886	**1.000**	0.947	**1.000**	8760	**7020**
R_2	0.844	**0.968**	0.928	**0.986**	**1404**	**1404**
R_3	0.682	**0.785**	0.861	**0.902**	2460	**2100**
R_4	0.444	**0.885**	0.833	**0.981**	**10 476**	**10 476**
R_5	**0.817**	**0.817**	**0.909**	**0.909**	**3690**	4710
R_6	0.801	**0.934**	0.833	**0.944**	**11 628**	**11 628**
R_7	0.282	**0.462**	0.635	**0.704**	**7448**	17 062
R_8	0.632	**1.000**	0.833	**1.000**	**396**	**396**
R_9	0.618	**0.695**	0.727	**0.773**	1134	**846**
R_{10}	0.189	**0.800**	0.600	**0.900**	**2756**	**2756**
R_{11}	**0.770**	**0.770**	**0.909**	**0.909**	**1764**	**1764**
R_{12}	0.158	**0.348**	0.702	**0.754**	**3820**	3980
R_{13}	**0.882**	0.768	**0.944**	0.889	**9438**	**9438**
R_{14}	**1.000**	**1.000**	**1.000**	**1.000**	152	**144**
R_{15}	**0.731**	0.704	**0.735**	0.706	**3444**	4088
R_{16}	0.118	**0.415**	0.571	**0.714**	**1464**	1692
R_{17}	**1.000**	**1.000**	**1.000**	**1.000**	**46 400**	**46 400**
R_{18}	0.733	**0.882**	0.900	**0.950**	**2640**	**2640**
R_{19}	0.301	**0.528**	0.701	**0.792**	**2640**	4384
R_{20}	0.863	**1.000**	0.900	**1.000**	630	**546**
R_{21}	0.364	**0.804**	0.769	**0.923**	**11 440**	**11 440**
R_{22}	**0.960**	0.950	**0.965**	0.957	9540	**6192**
R_{23}	0.557	**0.584**	**0.646**	0.636	33 156	**32 670**
R_{24}	0.782	**1.000**	0.905	**1.000**	**200**	**200**
R_{25}	0.519	**0.642**	0.639	**0.723**	**11 628**	11 844
R_{26}	0.917	**1.000**	0.957	**1.000**	**2624**	**2624**
R_{27}	0.757	**0.824**	0.778	**0.838**	**6402**	6468
R_{28}	0.920	**1.000**	0.944	**1.000**	**1664**	1742
Median	0.745	**0.820**	0.847	**0.916**	**3100**	4034
MAD	0.221	**0.155**	0.112	**0.097**	**6341**	6565
Max	**1.000**	**1.000**	**1.000**	**1.000**	**46 400**	**46 400**
Min	0.118	**0.348**	0.571	**0.636**	152	**144**

Table 6
Classification results for machine learning algorithms on real-world datasets. MAD stands for median absolute deviation, and Min and Max stand for minimum and maximum values, respectively. The best results are highlighted in bold.

ID	MCC					ACC				
	sDMN	MLP	RBFN	SVM	kNN	sDMN	MLP	RBFN	SVM	kNN
R_1	**1.000**	0.963	**1.000**	**1.000**	**1.000**	**1.000**	0.982	**1.000**	**1.000**	**1.000**
R_2	**0.968**	0.844	0.906	0.937	0.875	**0.986**	0.928	0.957	0.971	0.942
R_3	0.785	**0.975**	0.823	0.963	0.859	0.902	**0.988**	0.919	0.983	0.936
R_4	0.885	**1.000**	0.000	0.700	0.430	0.981	**1.000**	0.907	0.944	0.926
R_5	**0.817**	0.787	0.787	0.787	**0.817**	**0.909**	0.894	0.894	0.894	**0.909**
R_6	0.934	0.966	0.966	**1.000**	**1.000**	0.944	0.972	0.972	**1.000**	**1.000**
R_7	0.462	**0.582**	0.222	**0.582**	0.388	0.704	**0.791**	0.609	**0.791**	0.696
R_8	**1.000**	0.632	**1.000**	**1.000**	**1.000**	**1.000**	0.833	**1.000**	**1.000**	**1.000**
R_9	0.695	0.710	0.687	0.630	**0.763**	0.773	0.773	0.773	0.727	**0.818**
R_{10}	0.800	**0.875**	0.802	0.800	0.866	0.900	**0.933**	0.900	0.900	**0.933**
R_{11}	0.770	0.389	0.671	0.671	**1.000**	0.909	0.818	0.909	0.909	**1.000**
R_{12}	0.348	0.361	**0.377**	0.337	0.275	0.754	**0.772**	**0.772**	0.684	0.702
R_{13}	0.768	0.824	0.890	**1.000**	0.825	0.889	0.917	0.944	**1.000**	0.917
R_{14}	**1.000**	0.906	**1.000**	**1.000**	**1.000**	**1.000**	0.933	**1.000**	**1.000**	**1.000**
R_{15}	0.704	0.852	0.552	**0.880**	0.851	0.706	0.853	0.559	**0.882**	0.853
R_{16}	**0.415**	0.236	0.019	0.290	0.104	**0.714**	0.629	0.543	0.657	0.571
R_{17}	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
R_{18}	0.882	0.787	0.467	**1.000**	**1.000**	0.950	0.900	0.800	**1.000**	**1.000**
R_{19}	0.528	0.463	**0.622**	0.528	0.563	0.792	0.766	**0.831**	0.792	0.805
R_{20}	**1.000**	0.928	0.928	0.928	**1.000**	**1.000**	0.950	0.950	0.950	**1.000**
R_{21}	**0.804**	0.010	0.000	0.129	0.505	**0.923**	0.692	0.808	0.769	0.846
R_{22}	0.950	0.961	0.925	**0.985**	0.970	0.957	0.965	0.935	**0.987**	0.974
R_{23}	0.584	**0.679**	0.548	0.673	0.650	0.636	**0.749**	0.651	0.744	0.718
R_{24}	**1.000**	**1.000**	0.894	**1.000**	**1.000**	**1.000**	**1.000**	0.952	**1.000**	**1.000**
R_{25}	0.642	**0.888**	0.633	0.823	0.640	0.723	**0.916**	0.723	0.867	0.723
R_{26}	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
R_{27}	0.824	0.967	0.669	**0.989**	0.978	0.838	0.970	0.697	**0.990**	0.980
R_{28}	**1.000**	0.920	**1.000**	**1.000**	**1.000**	**1.000**	0.944	**1.000**	**1.000**	**1.000**
Median	0.820	0.863	0.794	**0.933**	0.870	0.916	0.922	0.908	**0.961**	0.939
MAD	**0.155**	0.204	0.252	0.197	0.205	0.097	**0.086**	0.119	0.092	0.099
Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Min	**0.348**	0.010	0.000	0.129	0.104	0.636	0.629	0.543	**0.657**	0.571

$p = 0.4942$, respectively. This finding suggests that sDMN is competitive with all four evaluated classifiers to solve a wide variety of classification problems.

In some classification problems, computational complexity is an important factor for obtaining a robust classification model in reasonable computing time. Also, the number of trainable parameters determines the complexity of the model. By considering both criteria, it is desirable to have robust classifiers with low time complexity and a small number of trainable parameters, which is "to do more with less" philosophy (Hernández et al., 2020). In this sense, Table 7 shows the number of trainable parameters (NTP) and each evaluated classifier's time complexity, except kNN. It is worth mentioning that both indices were measured on the final classification model obtained after the tuning process.

In general, MLP obtained the lowest NTP, although its computational complexity was the highest, which is because of the backpropagation algorithm run several epochs to adjust the network weights. Likewise, SVM required relatively few support vectors to define a classification model, although its computational complexity was the second highest since, despite using the SMO algorithm, the SVM evaluation dominates its computational complexity, which is quadratic. About sDMN, the NTP was the highest among its counterparts, although its time complexity is relatively low since LDCM has logarithmic complexity.

Because the classifiers presented different results for the same dataset, Fig. 12(a) compares the five methods around the average MCC of the group calculated for each dataset. For illustrative purposes, the kNN time complexity during predicting time is also shown. Notice that SVM, sDMN, and kNN tended to classify better than the group average, while RBFN is generally below this midpoint. It is notable that sDMN appears behind SVM, but requires less time complexity to train its parameters. Besides, SVM presented more considerable variations in time complexity than sDMN. Regarding kNN, it is well known that the time for searching the k-nearest neighbors increases directly proportional to the number of training samples and the k-value.

Finally, Fig. 12(b) shows the time complexity as a function of NTP. This graph reveals that NTP does not necessarily impact on the training time and vice versa. The training time depends on the algorithmic strategy to obtain the model parameters, whereas NTP depends on the structural elements of the model. For instance, sDMN obtained the highest NTP, since each hyperbox requires defining two parameters ($\mathbf{w}$ and $\mathbf{b}$) in $\mathbb{R}^d$, but its training time complexity is low due to LDCM's logarithmic complexity. In the SVM case, NTP is given by the number of support vectors, which are a subset of training patterns that define the SVM margins. However, finding such support vectors is computationally more expensive than finding the hyperboxes in sDMN because the algorithmic complexity of their respective training approaches is very different. This same behavior is observed in MLP, where the NTP was, in general, the lowest while the time complexity was the highest.

7. Conclusion and future work

In this paper, we proposed a modification to the dendrite morphological neuron (DMN) to generate smooth decision boundaries. The reasoning behind this idea is because DMN generates sharp and rough decision boundaries that do not accurately track

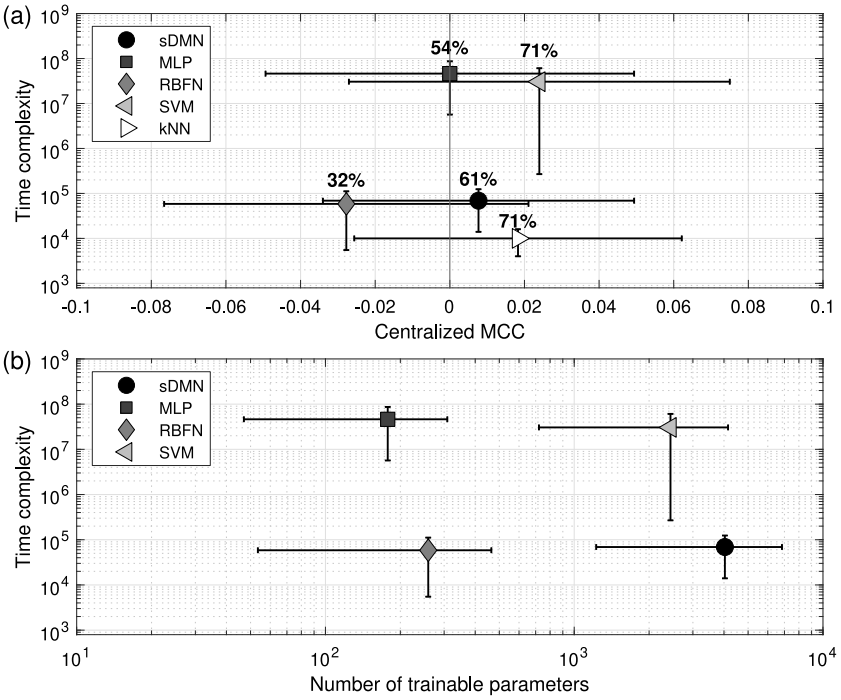

Fig. 12. (a) Time complexity versus the difference to the group MCC average calculated from each real-world dataset. Every point represents the median, and the error bars represent the median absolute deviation. Above each point is shown the percentage of datasets where the classifier performed better than the average group. (b) Time complexity as a function of the number of trainable parameters calculated from each real-world dataset. Every point represents the median, and the error bars represent the median absolute deviation.

the shape of class distributions. This feature is because the dendrites are represented by hyperboxes activated by the conventional maximum and minimum functions. Hence, the decision boundaries are forced to match the hyperboxes' faces.

In this study, the basic topology of DMN is preserved, and just the expressions for obtaining dendrites activations are modified by replacing the hard maximum and minimum functions by smooth versions. This modified dendritic model is called Smooth-DMN (sDMN). The proposed approach was evaluated on nine synthetic datasets and 28 real-world datasets. Furthermore, sDMN was compared against the conventional DMN model and four machine learning algorithms, specifically, two neural networks, Multilayer Perceptron (MLP) and Radial Basis Function Network (RBFN), Support Vector Machine with Gaussian kernel (SVM), and k-Nearest Neighbor (kNN) algorithm.

The sDMN results demonstrated that, as expected, using smooth minimum and maximum functions, the generalization capacity of DMN is improved. On average, sDMN obtained a classification performance 10% higher than the conventional DMN, with almost the same number of dendrites. Furthermore, the parameters of both dendritic models were trained with the Linear Divide and Conquer Method (LDCM), so their training time complexity is the same.

The so-called β hyperparameter controls the smoothness degree of smooth minimum and maximum functions, which should be adjusted adequately to cope with class distribution complexity. Besides, depending on the β value, sDMN can generate linear and nonlinear decision boundaries. This behavior was illustrated using two-dimensional synthetic datasets. In this study, we used the Differential Evolution algorithm to tune the β value as well as the hyperparameters of the LDCM algorithm, where the Matthews correlation coefficient (MCC) was the objective function. It is worth mentioning that we emphasized the use of MCC over the accuracy because MCC fairly quantifies the overall classification performance in unbalanced classes. Contrarily, the accuracy is biased to the success rate of the majority class; hence, it provides optimistic estimations.

The statistical comparison of sDMN with four machine learning algorithms revealed that all of them tended to classify the real-world datasets similarly. Nevertheless, SVM reached superior values of MCC and accuracy, which confirms its well-demonstrated generalization capabilities. Because classifiers presented different results for the same dataset, their classification results were compared against the group average. This analysis showed that SVM, kNN, and sDMN generally tended to classify better than their counterparts for most datasets, while RBFN was the classifier with the lowest classification performance.

The sDMN model obtained comparable classification results concerning MLP and SVM, although both classifiers required, on average, 95% less trainable parameters than sDMN. Contrarily, the computational cost for training an sDMN was, on average, 99% lower than MLP and SVM. Therefore, obtaining models with high classification performance, low training time, and simple architecture is an open problem that deserves further research.

It is worth mentioning that the DE algorithm was useful to tune the sDMN hyperparameters (m, ϵ, and β) in a reasonable computation time. Nevertheless, other search procedures can be used, although the computational cost should be considered. For instance, the grid search method is simple to implement, but it presents polynomial time complexity as a function of the number of variables to optimize. Besides, it requires defining a constrained search space in terms of the grid's granularity and upper and lower boundaries, impacting the exploration of feasible solutions, while DE is unconstrained to explore the search space.

In the context of simplifying the sDMN architecture, recently, we presented a preliminary study for reducing the number of parameters of the DMN model by using a dendrite representation based on hyperspheres (Gómez-Flores & Sossa, 2020). In this way, the number of parameters could be reduced to about 50% since a hypersphere requires defining a centroid and a radius in $\mathbb{R}^d$. Hence, the number of parameters decreases from $2d$ to $d + 1$.

Table 7
Number of trainable parameters and training time complexity for machine learning algorithms on real-world datasets. MAD stands for median absolute deviation, and Min and Max stand for minimum and maximum values, respectively. The best results are highlighted in bold.

	Trainable parameters				Time complexity ($\times 10^3$)			
ID	sDMN	MLP	RBFN	SVM	sDMN	MLP	RBFN	SVM
R_1	7020	**464**	728	2340	**214**	229 376	338	235 930
R_2	1404	74	**50**	2313	69	40 524	**22**	30 537
R_3	2100	422	**378**	2826	**123**	590 900	318	87 049
R_4	10 476	128	**44**	2250	115	58 320	**18**	76 528
R_5	4710	**38**	110	5340	115	19 958	**53**	77 528
R_6	11 628	**170**	498	6698	147	51 520	**132**	119 859
R_7	17 062	**530**	662	10 640	**281**	522 144	591	387 460
R_8	396	**26**	**26**	306	4	1210	**1**	245
R_9	846	**166**	**166**	1278	19	28 800	**18**	2986
R_{10}	2756	194	**66**	2184	41	48 060	**14**	12 048
R_{11}	1764	**86**	212	954	20	8080	**18**	3305
R_{12}	3980	**54**	288	2500	**64**	25 056	115	27 248
R_{13}	9438	**290**	**290**	7095	139	88 200	**83**	108 056
R_{14}	144	**19**	**19**	128	5	1890	**1**	292
R_{15}	4088	**300**	660	3444	**52**	80 784	66	18 353
R_{16}	1692	**74**	92	1272	20	19 584	**18**	3371
R_{17}	46 400	**186**	1788	31 200	**607**	87 472	796	1 580 858
R_{18}	2640	**252**	**252**	2376	46	42 000	**39**	14 823
R_{19}	4384	**200**	266	2920	**69**	124 380	133	30 559
R_{20}	546	**25**	47	315	13	3580	**5**	1570
R_{21}	11 440	**190**	284	3300	142	44 344	**64**	112 445
R_{22}	6192	**995**	1203	4122	**569**	1 975 050	1724	1 400 406
R_{23}	32 670	1477	**1407**	25 218	**732**	2 493 288	1888	2 222 368
R_{24}	200	**39**	57	85	10	6208	**6**	941
R_{25}	11 844	**556**	648	6084	**189**	402 864	385	188 623
R_{26}	2624	**40**	230	400	**39**	7524	40	11 182
R_{27}	6468	**655**	701	7612	**130**	548 856	298	96 060
R_{28}	1742	**37**	71	689	23	5120	**8**	4326
Median	4034	**178**	259	2438	69	46 202	**59**	30 548
MAD	6565	**230**	334	4362	**124**	351 011	306	329 415
Max	46 400	**1477**	1788	31 200	**732**	2 493 288	1888	2 222 368
Min	144	**19**	**19**	85	4	1210	**1**	245

Future work involves including the smooth minimum and maximum function in the spherical-shaped dendrite model. Also, it is necessary to consider the undesirable properties of hyperspheres in high dimensions, where the volume of a hypersphere tends to zero when dimensionality grows. Therefore, the spherical-shaped dendrites can be defined in some subspace of the input space to avoid such undesirable effects (G., 2012, 2017).

Because the smooth minimum and maximum functions are differentiable, one research direction is to explore methods based on gradient descent to train the parameters of an sDMN. Note that first and second-order derivatives given in the Appendix are useful for this purpose. Furthermore, sDMN can be extended to form neural networks and even include them in recently developed hybrid methods aiming to improve their classification performance. Besides, sDMN can be potentially adapted to solve regression problems.

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.

Acknowledgments

We want to express our sincere appreciation to the Instituto Politécnico Nacional for the support provided to carry out this work. This research was supported Fondo SEP-Cinvestav de Apoyo a la Investigación FidSC2018/145, Instituto Politécnico Nacional SIP-IPN 20200630, CONACYT Fronteras de la Ciencia 65, and FORDECYT-PRONACES 6005 grants.

Appendix. Derivatives of the smax function

The smooth maximum of the array $\mathbf{u} = [u_1, \ldots, u_k]$ with k numbers is given by Takacs (2010)

$$\operatorname{smax}(\mathbf{u}) = \sum_{i=1}^{k} p_i u_i, \tag{A.1}$$

where p_i can be interpreted as a measure of dominance of the ith number over the others, which is expressed as:

$$p_i = \frac{\exp(\beta u_i)}{\sum_{j=1}^{k} \exp(\beta u_j)}, \quad \beta > 0, \tag{A.2}$$

where the parameter β controls the degree of smoothness.

The first order partial derivative of smax (for $i = 1, \ldots, k$) is:

$$\frac{\partial \operatorname{smax}(\mathbf{u})}{\partial u_i} = p_i \left(1 + \beta \left(u_i - \operatorname{smax}(\mathbf{u})\right)\right). \tag{A.3}$$

The second order partial derivative of smax (for $i = 1, \ldots, k$) is:

$$\frac{\partial^2 \operatorname{smax}(\mathbf{u})}{\partial u_i \partial u_j} = \left(-p_i s'_j - p_j s'_i + \delta_{ij}\left(s'_i + p_i\right)\right)\beta, \tag{A.4}$$

where $\delta_{ij} = [i = j]$ is the Kronecker delta and s'_i is given by Eq. (A.3).

References

Araujo, R. (2012). A morphological perceptron with gradient-based learning for brazilian stock market forecasting. *Neural Networks*, *28*, 61–81.

Araujo, R., Oliveria, A. L. I., & Meira, S. (2013). A swarm-based evolutionary morphological approach for binary classification problems. *Journal of the Brazilian Computational Intelligence Society*, *11*, 48–55.

Arce, F., Zamora, E., Focil-Arias, C., & Sossa, H. (2019). Dendrite ellipsoidal neurons based on k-means optimization. *Evolving Systems*, *10*, 381–396.

Arce, F., Zamora, E., & Sossa, H. (2017). Dendrite ellipsoidal neuron. In *2017 International joint conference on neural networks* (pp. 795–802).

Arce, F., Zamora, E., Sossa, H., & Barrón, R. (2018). Differential evolution training algorithm for dendrite morphological neural networks. *Applied Soft Computing*, *68*, 303–313.

Barmpoutis, A., & Ritter, G. X. (2007). *Orthonormal basis lattice neural networks* (pp. 45–58). Berlin, Heidelberg: Springer Berlin Heidelberg.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*, 27:1–27:27.

Das, S., Abraham, A., & Konar, A. (2008). Automatic clustering using an improved differential evolution algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, *38*, 218–237.

Davidson, J., & Hummer, F. (1993). Morphology neural networks: An introduction with applications. *Circuits Systems and Signal Process*, *12*, 177–210.

Dua, D., & Graff, C. (2017). UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Duda, R., Hart, P., & Stork, D. (2012). *Pattern classification* (2nd ed.). John Wiley & Sons.

G., Dudek (2012). An artificial immune system for classification with local feature selection. *IEEE Transactions on Evolutionary Computation*, *16*, 847–860.

G., Dudek (2017). Artificial immune system with local feature selection for short-term load forecasting. *IEEE Transactions on Evolutionary Computation*, *21*, 116–130.

Gómez-Flores, W. (2020). Datasets for machine learning. URL: https://github.com/wgomezf/datasets-for-machine-learning.

Gómez-Flores, W., & Sossa, H. (2020). Towards dendrite spherical neurons for pattern classification. In K. Figueroa Mora, J. Anzurez Marín, J. Cerda, J. Carrasco-Ochoa, J. Martínez-Trinidad, & J. Olvera-López (Eds.), *MCPR 2020: Pattern recognition* (pp. 14–24). Cham: Springer.

Gorodkin, J. (2004). Comparing two k-category assignments by a k-category correlation coefficient. *Computational Biology and Chemistry*, *28*, 367–374.

Hernández, G., Zamora, E., Sossa, H., & Téllez, F. (2020). Hybrid neural networks for big data classification. *Neurocomputing*, *390*, 327–340.

Minsky, M., & Papert, S. (1969). *Perceptrons. An introduction to computational geometry*. MIT Press.

Mondal, R., Santra, S., & Chanda, B. (2019). Dense morphological network: An universal function approximator. arXiv:1901.00109.

Pessoa, L. F., & Maragos, P. (2000). Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition. *Pattern Recognition*, *33*, 945–960.

Price, K., Storn, R., & Lampinen, J. (2005). *Natural computing series*, *Differential evolution: A practical approach to global optimization*. Springer.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge University Press.

Ritter, G. X., & Schmalz, M. S. (2006). Learning in lattice neural networks that employ dendritic computing. In *2006 IEEE international conference on fuzzy systems* (pp. 7–13).

Ritter, G. X., & Sussner, P. (1996). An introduction to morphological neural networks. In *Proceedings of 13th international conference on pattern recognition* (pp. 709–717).

Ritter, G. X., & Urcid, G. (2003). Lattice algebra approach to single-neuron computation. *IEEE Transactions on Neural Networks*, *14*, 282–295.

Ritter, G. X., Urcid, G., & Juan-Carlos, V. (2014). Two lattice metrics dendritic computing for pattern recognition. In *2014 IEEE international conference on fuzzy systems* (pp. 45–52).

Sossa, H., Arce, F., Zamora, E., & Guevara, E. (2018). *Morphological neural networks with dendritic processing for pattern classification* (pp. 27–47). Cham: Springer.

Sossa, H., & Guevara, E. (2014). Efficient training for dendrite morphological neural networks. *Neurocomputing*, *131*, 132–142.

Sprent, P., & Smeeton, N. C. (2001). *Applied nonparametric statistics methods* (3rd ed.). CRC Press.

Sussner, P. (1998). Morphological perceptron learning. In *Proceedings of the 1998 IEEE international symposium on intelligent control held jointly with IEEE international symposium on computational intelligence in robotics and automation intell* (pp. 477–482).

Sussner, P., & Campiotti, I. (2020). Extreme learning machine for a new hybrid morphological/linear perceptron. *Neural Networks*, *123*, 288–298.

Sussner, P., & Esmi, E. L. (2011). Morphological perceptrons with competitive learning: Lattice-theoretical framework and constructive learning algorithm. *Information Sciences*, *181*, 1929–1950.

Takacs, G. (2010). Smooth maximum based algorithms for classification, regression, and collaborative filtering. *Acta Technica Jaurinensis*, *3*, 27–63.

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern recognition* (4th ed.). Burlington, MA: Academic Press.

Zamora, E., & Sossa, H. (2017). Dendrite morphological neurons trained by stochastic gradient descent. *Neurocomputing*, *260*, 420–431.