Applied Soft Computing 68 (2018) 303–313

Contents lists available at ScienceDirect

Applied Soft Computing

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

Differential evolution training algorithm for dendrite morphological neural networks

Fernando Arce [a,*], Erik Zamora [b], Humberto Sossa [a], Ricardo Barrón [a]

[a] Instituto Politécnico Nacional, CIC, Av. Juan de Dios Batiz S/N, Col. Nueva Industrial Vallejo, Gustavo A. Madero, 07738 Ciudad de México, Mexico
[b] Instituto Politécnico Nacional, UPIITA, Av. Instituto Politécnico Nacional 2580, Col. Barrio la Laguna Ticoman, 07340 Ciudad de México, Mexico

ARTICLE INFO

Article history:
Received 27 December 2016
Received in revised form 15 January 2018
Accepted 20 March 2018
Available online 3 April 2018

Keywords:
Dendrite morphological neural network
Differential evolution
Hyper-box
Machine learning
Morphological neural network

ABSTRACT

Dendrite morphological neural networks are emerging as an attractive alternative for pattern classification, providing competitive results with other classification methods. A key problem in the design of these neural networks is the election of the number of their dendrites. Most training methods are heuristics that do not optimize the learning parameters. Therefore, we propose a new training algorithm for classification tasks based on an optimization approach: differential evolution. We show that the besought method increases classification performance and also optimizes the number of dendrites. For generating the initial population of hyper-boxes, we adopt two techniques: one based on the division of an initial hyper-box, and the other on an initial clustering using the so-called k-means++. Both alternatives were tested on four synthetic and 11 real databases as benchmarks overcoming the state-of-the-art morphological neuron training methods as well as the radial basis networks. The proposed training algorithm achieved a favorable average accuracy compared with the well-known multilayer perceptrons and support vector machines. In addition, a real-life problem was solved by this method to recognize geometric figures using a Nao robot.

1. Introduction

The fundamental purpose of pattern classification is to learn a mathematical model that relates an input pattern $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$ to its class $C^1, C^2, \ldots C^j$. This relationship must be as robust as possible to irrelevant variations in the patterns, and must be able to capture the fundamental relationship between data, assuming that there are invariant features in the data. Many types of neural networks have been proposed for this task and examined in computer science literature, over more than 60 years. One of the most successful has been the group of classical perceptrons which use an affine transformation and a non-linear activation function. These perceptrons dichotomize the input space using a hyper-plane determined by synaptic weights and biases. Thus, single-layer perceptrons can only solve classification problems with linear separability. This work examines another type of neuron which has been little studied in literature, that is, the neuron with morphological dendrites. The neuron solves linear and non-linear classification problems using a single layer. The morphological processing involves min or max operations; these operators generate the piecewise boundaries for classification problems, and have the advantage of being implemented easily in logic devices.

This paper is an extension of a work recently reported in the SSCI 2016 congress [1] and focuses on a specific type of morphological neural networks (MNN) which is known as a dendrite morphological neural network (DMNN). Each dendrite represents a hyper-box, and these hyper-boxes group classes. A hyper-box is a generalization in a high dimensional space from a 2D box. These dendrites have played an important role in previously proposed training methods. The most common approach is to enclose patterns with hyper-boxes and label each with the correct class, and many training methods have been published based on this approach. Most of these are heuristic, and are not based on the optimization of learning parameters. In this paper, we show that a parameter optimization can be computed, and this can improve the classification performance of DMNN over solely heuristic-based and other popular machine learning methods.

On the other hand, differential evolution (DE) is recognized as a powerful optimization technique [2–4]. This method is especially useful for problems with cost functions that are non-differentiable, non-continuous, non-linear, multi-dimensional or which have many local minima. The main advantage is that the method does not require the calculation of analytical expressions such as gra-

* Corresponding author.
E-mail addresses: fernando.arce.vega@gmail.com, b150689@sagitario.cic.ipn.mx (F. Arce).

https://doi.org/10.1016/j.asoc.2018.03.033
1568-4946/

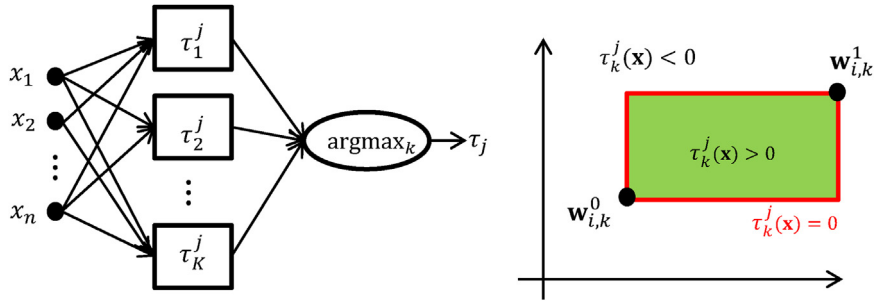

Fig. 1. Typical DMNN. Left: Architecture of a DMNN with a single output neuron. Right: An example of a hyper-box in 2D generated by its dendrite weights.

dients. Furthermore, the method is robust in finding the global optimization in comparison with gradient-based methods.

In this paper, we adapt the DE technique to find the number of dendrites and their dendrite parameters for classification problems. This is useful since a DMNN has a cost function that is non-continuous due to the argmax operator at the neuron output.

When classical perceptrons are grouped into layers of connected neurons in series, the hidden layers try to extract invariant features as far as possible, and the output layer is devoted to classifying such features. Thus, a multilayer model with classical perceptrons can solve non-linear classification problems. This multilayered strategy is exploited by a variety of other models: multilayer perceptrons (MLP) [5], radial basis network (RBN) [6] and support vector machine (SVM) [7] with kernel trick. In all these models, there is at least one layer for feature extraction. We show that a DMNN is able to achieve similar or better performance than these multilayer models without a feature extraction layer. The contributions of this paper are: (1) to the best of our knowledge, a DMNN is trained by an evolutionary algorithm for the first time; and (2) our proposal outperforms the most effective heuristic-based and other popular machine learning methods.

The remaining sections are organized as follows: Section 2 introduces the computational basis of the DMNN. Section 3 describes the previous work in the area of MNN. Section 4 presents the proposed evolutionary algorithm. Section 5 discusses the results of experiments to assess the effectiveness of our proposal. Finally, Section 6 gives our conclusions and proposed future work.

2. Computational basis for DMNN

DMNN computations are based on lattice algebra. More detailed information can be found in [8,9].

The morphological processing involves min ($\wedge$) or max ($\vee$) operations; these operators generate piecewise boundaries for classification problems, and have the advantage of being easily implemented in logic devices because computational processing is based on comparative operators instead of products.

In this research, we used a fully connected DMNN with a single output neuron, where each incoming pattern is first processed for all the dendrites, and the dendrite with the biggest value is then selected. Fig. 1 shows the architecture of a DMNN with a single output neuron and an example of a hyper-box generated by its dendrite weights in 2D.

The computation performed τ_k^j by the kth dendrite for the jth class can be expressed by Eq. (1):

$$\tau_k^j = \wedge_{i=1}^{n}(x_i + w_{ik}^1) \wedge -(x_i + w_{ik}^0), \tag{1}$$

where x_i is an input vector, n is the vector dimensionality, $i \in I$ and $I \in \{1, \ldots, n\}$ represents the set of all input neurons with terminal fibers that synapse on the kth dendrite. w_{ik}^0 and w_{ik}^1 are the synaptic weights corresponding to the set of terminal fibers of the ith neuron that synapse on the kth dendrite; w_{ik}^1 is the activation terminal fiber and w_{ik}^0 is the inhibition terminal fiber.

If $\tau_k^j > 0$, x is inside the hyper-box; if $\tau_k^j = 0$, x is over the hyper-box boundary; and if $\tau_k^j < 0$, x is outside the hyper-box.

On the other hand, the output value of a DMNN with a single output neuron τ_j is the maximum argument calculated in Eq. (2):

$$\tau_j = argmax_k(\tau_k^j), \tag{2}$$

where the $argmax_k$ function selects only one dendrite from all network dendrites, generating a scalar. This argmax function enables the DMNN classifying patterns that are outside the hyper-boxes and builds more complex decision boundaries than simple hyper-boxes. If Eq. (2) produces more than one output, argmax selects the first maximum argument as an index class.

In order to explain the dendrite computations of a DMNN, Fig. 2 (Left) displays two hyper-boxes which could be generated by any DMNN training algorithm and cover all the patterns. Blue dot points belong to class one C^1 and green cross points to class two C^2. Fig. 2 (Right) presents the DMNN formed based on these two hyper-boxes. The input layer is connected to the output neuron via the dendrites. The geometrical calculation explanation executed by the dendrites is that each of these determines a hyper-box which can be represented by its weight values w_{ij}.

For testing the generated network, two noisy patterns were proposed; $\tilde{X}_1 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$ of class C^1 and $\tilde{X}_2 = \begin{pmatrix} 7 \\ 3 \end{pmatrix}$ of class C^2. The dendrite computations are:

$$\tau_1^1(\tilde{X}_1) = \tau_1^1\begin{pmatrix} 3 \\ 0 \end{pmatrix} = [(3-1) \wedge -(3-5)] \wedge [(0-1) \wedge -(0-5)]$$
$$= [2 \wedge -1] = -1$$

$$\tau_1^2(\tilde{X}_1) = \tau_1^2\begin{pmatrix} 3 \\ 0 \end{pmatrix} = [(3-4) \wedge -(3-8)] \wedge [(0-4) \wedge -(0-8)]$$
$$= [-1 \wedge -4] = -4$$

$$\tau_1^1(\tilde{X}_2) = \tau_1^1\begin{pmatrix} 7 \\ 3 \end{pmatrix} = [(7-1) \wedge -(7-5)] \wedge [(3-1) \wedge -(3-5)]$$
$$= [-2 \wedge 2] = -2$$

$$\tau_1^2(\tilde{X}_2) = \tau_1^2\begin{pmatrix} 7 \\ 3 \end{pmatrix} = [(7-4) \wedge -(7-8)] \wedge [(3-4) \wedge -(3-8)]$$
$$= [1 \wedge -1] = -1$$

And applying the argmax function:

$$\tau = \tau_1^1(\tilde{X}_1) \vee \tau_1^2(\tilde{X}_1) = -1 \vee -4 = -1$$

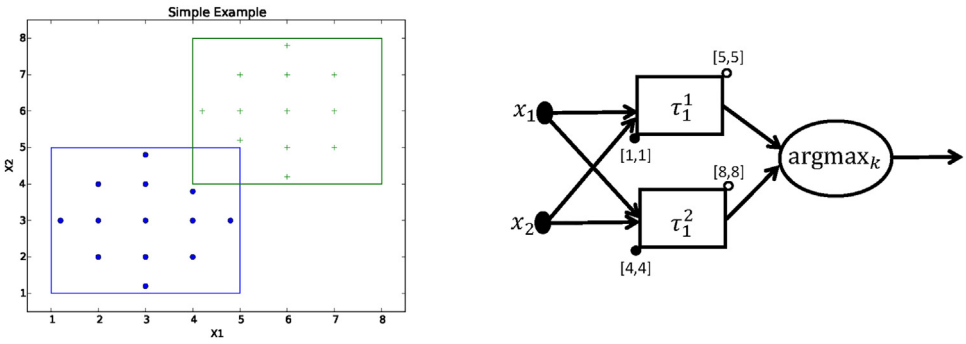

Fig. 2. Simple example of a DMNN. Left: Two hyper-boxes which cover all the patterns. Right: Formed DMNN based on these two hyper-boxes (black circles denote excitatory connections and white circles inhibitory connections).

$$\tau = \tau_1^1(\tilde{X}_2) \vee \tau_1^2(\tilde{X}_2) = -2 \vee -1 = -1$$

Therefore, the neuron maps out $\tilde{X}_1$ to C^1 and $\tilde{X}_2$ to C^2 correctly.

3. Previous work

As a combination of image algebra and artificial neural networks, MNN were proposed by Ritter and Wilson [10,11]. In this work, they described the way in which image algebra is able to compute the operations performed by the typical neural networks. After that, Davidson et al. [12,13] used an MNN to determine the template output for grayscale images, and described several applications within image processing.

In 1996, Ritter and Sussner [14] demonstrated that MNN was capable of solving the NAND problem, and improved it by proposing the single-layer morphological perceptrons (SLMP) with a training method. They used the MNN to form associative memories in [15].

Sussner [16] improved the SLMP by introducing the multilayer morphological perceptrons (MLMP) in 1998, which was capable of solving the XOR problem using four morphological neurons in the hidden layer. It was implemented with a supervised learning algorithm.

Pessoa and Maragos [17] adapted the back propagation algorithm to a morphological/rank/linear filter (MRL). In this work, they combined typical perceptrons with MRL filters and obtained similar or better performance than using MLP in classifying handwritten digits. A substantial difference between the existing MNNs and this method was that it did not use hyper-boxes to establish a decision boundary, and MRL thus created more complex decision boundaries.

Due to advances in the biophysics of computations in 2003, Ritter and Urcid [15] enhanced the SLMP by providing the network with dendral structures. This network produced connected non-convex regions performed with lattice computations and had an automatic training algorithm.

In 2006, Barmpoutis and Ritter [18] upgraded the dendral network [15] by changing the orthonormal basis of each neural dendrite. Here, the resultant hyper-boxes were able to take a different orientation of their coordinate axes.

After Ritter and Urcid enhanced the SLMP [15], Ritter et al. [9,19] generalized the SLMP by proposing the single-layer lattice perceptron (SLLP) to handle non-linear multiclass problems, and created two training algorithms, the merging and the elimination method.

Sussner and Esmi [20] put forward the morphological perceptron (MP), with competitive output neurons and a training algorithm which increases the number of hidden neurons automatically.

Following the MP improvement, Sussner et al. [21,20,22] conditioned the MP with competitive neurons, by providing the argmax function in the network output.

In 2012, Araujo [23] implemented the gradient steepest descent method with a morphological perceptron to resolve time series such as the stock market forecasting problem. He called this network increasing morphological perceptron (IMP), and was trained by back propagation.

In 2014, Sossa et al. [24] proposed an efficient training algorithm for DMNN which iteratively divides the search space using hyper-boxes until all the patterns are well assigned in the training phase. This algorithm outperforms the Ritter algorithm [15] in all the examined datasets. However, this method has a disadvantage in terms of overfitting, since the training phase always obtains an error of zero percent.

In the same year, Ritter et al. [25] added two lattice metrics, L_1 and L_∞, to the SLLP to form more complex decision boundaries, known as polytopes. This network uses an extension of the elimination training algorithm proposed in [9,19].

Sossa et al. [26] adapted a DMNN for 3D object classification and one year after modified the architecture of a typical RBN to automatically set the hyper-parameters [27]. The first step in this method is to apply the algorithm reported in [24] and then to take the center and the dimensions of each hyper-box and apply these to a Gaussian kernel.

Then, Sossa and Zamora [28] modified a DMNN architecture by adding a softmax layer to train the network based on stochastic gradient descent for classifications problems.

Most of these methods use heuristics to determine the location of the hyper-boxes in the input space. In contrast, we propose carrying out a search of those that were best classified based on an evolutionary approach. This is not the first time that parameter optimization has been suggested in the context of morphological neurons. As mentioned earlier, Pessao et al. [17] and Araujo [23] have investigated gradient-based optimization for certain architectures of morphological neurons.

The main advantage of an optimization based on evolution is that the analytical evaluation of expressions such as gradients need not be carried out, and discontinuities of the cost function are not necessary.

4. The proposed training algorithm

This section explains the steps involved in the new training algorithm. Firstly, however, two definitions are provided.

Definition 1. Let $x_1, x_2, \ldots, x_m$ be a finite set of sample patterns, where each pattern $x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in \mathbb{R}^n$ for $i = 1, 2, \ldots, m$ is an n-dimensional vector. Furthermore, each pattern x_i belongs to only one C^j class, for $j = 1, 2, \ldots, p$, where $p > 1$ is a finite number.

Definition 2. A hyper-box HB^n is an n-dimensional box that contains a finite set of patterns $x \in \mathbb{R}^n$. This HB^n defines the weights w_i^l, $l = \{0, 1\}$ where 1 denotes excitation and 0 inhibition.

Eq. (3) defines a hyper-box:

$$HB_k^n = \{x \in \mathbb{R}^n : w_{ik}^1 \leq x_i \leq w_{ik}^0, i = 1, \ldots, n\}, \quad (3)$$

The purpose of our training algorithm is to create and place a set of hyper-boxes $HB_k^n \in \mathbb{R}^{k \times 2n}$ for $k \in \{1, 2, \ldots, K\}$ that establish an optimum decision boundary between the classes using the smallest number.

4.1. Training algorithm

This sub-section provides the steps of the proposed algorithm, which uses two initialization methods for the hyper-boxes.

- Hyper-boxes division (HBd) initialization:

1. For each class C^j, select all the patterns and open one hyper-box per class, with a length such that all the patterns of each class remain inside. Sussner and Laureano [22] proposed a similar algorithm which keeps dividing these in order to avoid overlapping; however, the algorithm here suggested does not need to take into account the overlapping of the hyper-boxes.
2. Divide each hyper-box into smaller hyper-boxes along the first axis on equal terms by a factor d, for $d \in \mathbb{Z}^+$ until q divisions have been carried out.

- K-means++ initialization:

1. For each class, generate q clusters by using the clustering algorithm k-means++ [29]. These clusters are transformed into hyper-boxes.
2. Calculate a local error in the DMNN based on the generated hyper-boxes and in the actual class. If necessary, increment the number of clusters, q, for each class.
3. Continue with the next classes.

- Apply DE, as in [30], to the actual hyper-boxes.
- Select the best set of hyper-boxes that produce the smallest error. Algorithm 1 shows the pseudo-code of DE applied to a DMNN.

Algorithm 1. Pseudo-code of DE applied to DMNN

```
Begin
Generate initial population of solutions.
for d = 1 to q
  Repeat:
For the entire population, calculate the fitness value.
For each parent, select two solutions at random and the best parent.
Create one offspring using DE operators.
If the offspring is better than the parents:
  Parent is replaced by the offspring.
  Until a stop condition is satisfied.
End
```

a) **Initialization**: The set of hyper-boxes forms a $k \times n$ matrix called *ListH*. Each row of *ListH* forms the corners of a hyper-box and represents a chromosome. The initial population of parents is formed by multiplying t random vectors F per *ListH*, where t is the population size. The multiplication of one vector F per *ListH* conforms a parent of the total population, and each multiplication changes the original size and position of each hyper-box. In this case, DE has preliminary knowledge about the solution space.

b) **Fitness Function**: Based on the corners of the generated hyper-boxes that enclose the patterns for the C^j classes, minimize the classification error, which is calculated using Eq. (4), dividing the number of misclassified patterns, g, by the total number of patterns, m:

$$\%error = \frac{g}{m} \times 100, \quad (4)$$

c) **Mutation**: Select uniformly at random one from each ten parents, and permute 30% of the k rows with the purpose of changing the class order. These parameters were chosen based on preliminary experiments.

d) **Crossover**: We do not report crossover, since the results obtained both with and without this are similar.

e) **Offspring creation using DE operators**: Create the offspring with the actual parents using a DE operator, as in Eq. (5):

$$ListH_t = ListH_a + F \times (ListH_b - ListH_c), \quad (5)$$

where a, b and c must be distinct. $ListH_t$ is the offspring, F is a random vector, $ListH_b$ and $ListH_c$ are randomly chosen parents and $ListH_a$ is the best member of the population, individual duplication is not accepted.

f) **Stop condition**: DE stops if the error criteria is satisfied or the full number of iterations, q, has been reached. The algorithm applies the mutation mechanism to avoid local minima. It is worth mentioning that this does not guarantee solving the problem because it is not a convex function.

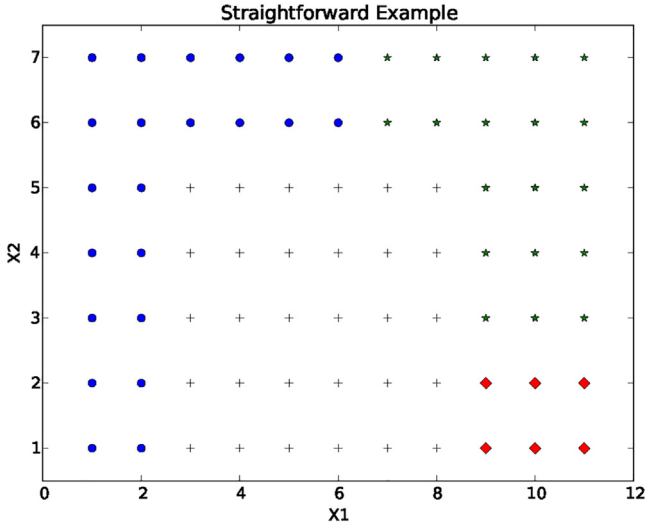

Fig. 3. Simple example of four classes with two attributes. The points in C^1 are shown as blue dots, points in C^2 as black crosses, points in C^3 as green stars and points in C^4 as red diamonds. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

4.2. HBd initialization method

One of the most important problems involved in the DMNN training algorithm is the election of the dendrite number and the optimization of the synaptic weights for each dendrite.

In order to explain this training algorithm with HBd initialization, a straightforward example of four classes with two features is presented. Fig. 3 illustrates the problem to be solved; the blue dots belong to C^1, black crosses to C^2, green stars to C^3 and red diamonds to C^4.

1) Select all the patterns of each class and open a hyper-box for each class. Fig. 4 shows the hyper-boxes that cover all the patterns.
2) Divide each hyper-box by a factor d and apply DE to the resultant hyper-boxes in order to set them in the best position with the best size. Fig. 5 illustrates the divisions of each hyper-box

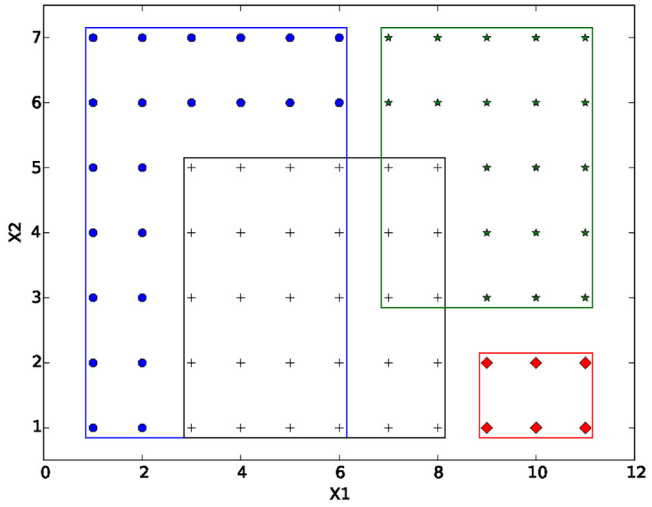

Fig. 4. Step 1: Patterns of each C^l class enclosed with a hyper-box for each class. The blue HB^1 encloses all patterns from C^1, the black HB^2 encloses all patterns from C^2, the green HB^3 encloses all patterns from C^3 and the red HB^4 encloses all patterns from C^4. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

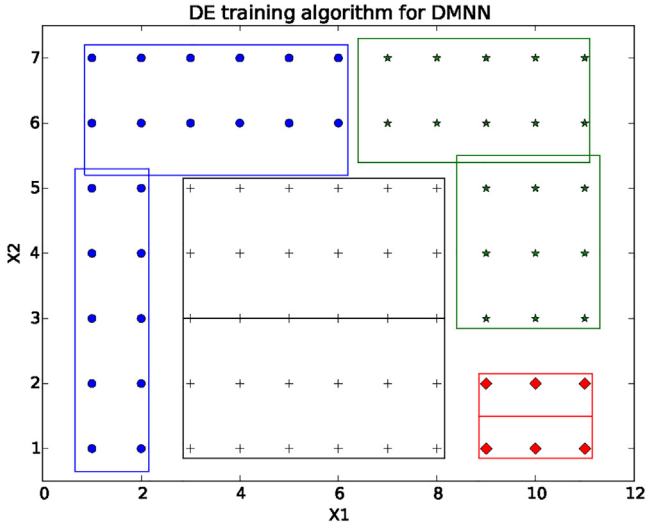

Fig. 6. Step 3: Selection of the best set of hyper-boxes generated by the training algorithm.

generated in step 2 when $d = 1$ and $d = 2$. When $d = 1$ there is no division.

3) Select the best set of hyper-boxes. Fig. 6 displays the best set generated after applying the training algorithm, with $d = 2$.

4.3. K-means++ initialization method

K-means is a broadly used machine-learning algorithm that groups patterns to their nearest clusters. In this work, we decided to apply k-means++ which is an improved version of k-means that increases the speed and accuracy. In k-means++, the initial centroid assignation is computed by means of a probability distribution which gives preference to further patterns from the previous chosen centroids. This is unlike the classical k-means which assigns initial centroids in an arbitrary way [29].

The same example of four classes with two features is used to illustrate the k-means++ method for initialization.

1) For C^1, form one hyper-box using the clustering algorithm k-means++ and calculate the local error, and iterate until $q = 2$. Fig. 7 (a) presents the hyper-boxes generated when $q = 1$ and $q = 2$, for C^1.
2) Continue with C^2, C^3 and C^4. Fig. 7 (b) exhibits the hyper-boxes generated for C^2 and Fig. 7 (c) shows those for C^3 and C^4. Finally, Fig. 8 displays all hyper-boxes generated using the k-means++ initialization method for this straightforward example.

For this simple dataset, it was not necessary to apply DE to the generated hyper-boxes, since the k-means++ initialization method obtained a final error of zero. It appears that the k-means++ initialization method places the hyper-boxes more effectively than the HBd initialization method; however, in practical experiments better performance was obtained using the HBd initialization method.

5. Experiments

Experimental validation results are presented in this section. Experiments were performed using four synthetic datasets, 11 datasets from the UCI machine learning repository [31] and one dataset focused on classifying geometric figures using a Nao robot. On the basis of these results, the new training algorithm demon-

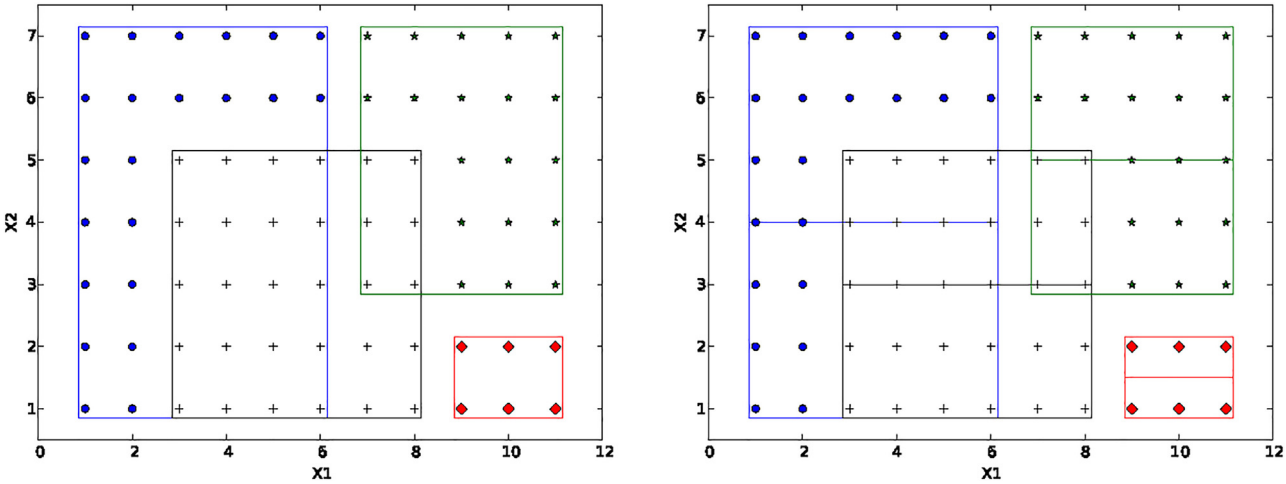

Fig. 5. Step 2: Divide each hyper-box by a factor d and apply DE to the resultant hyper-boxes. Left: When d is equal to 1, there are four hyper-boxes. Right: When d is equal to 2, there are eight hyper-boxes.

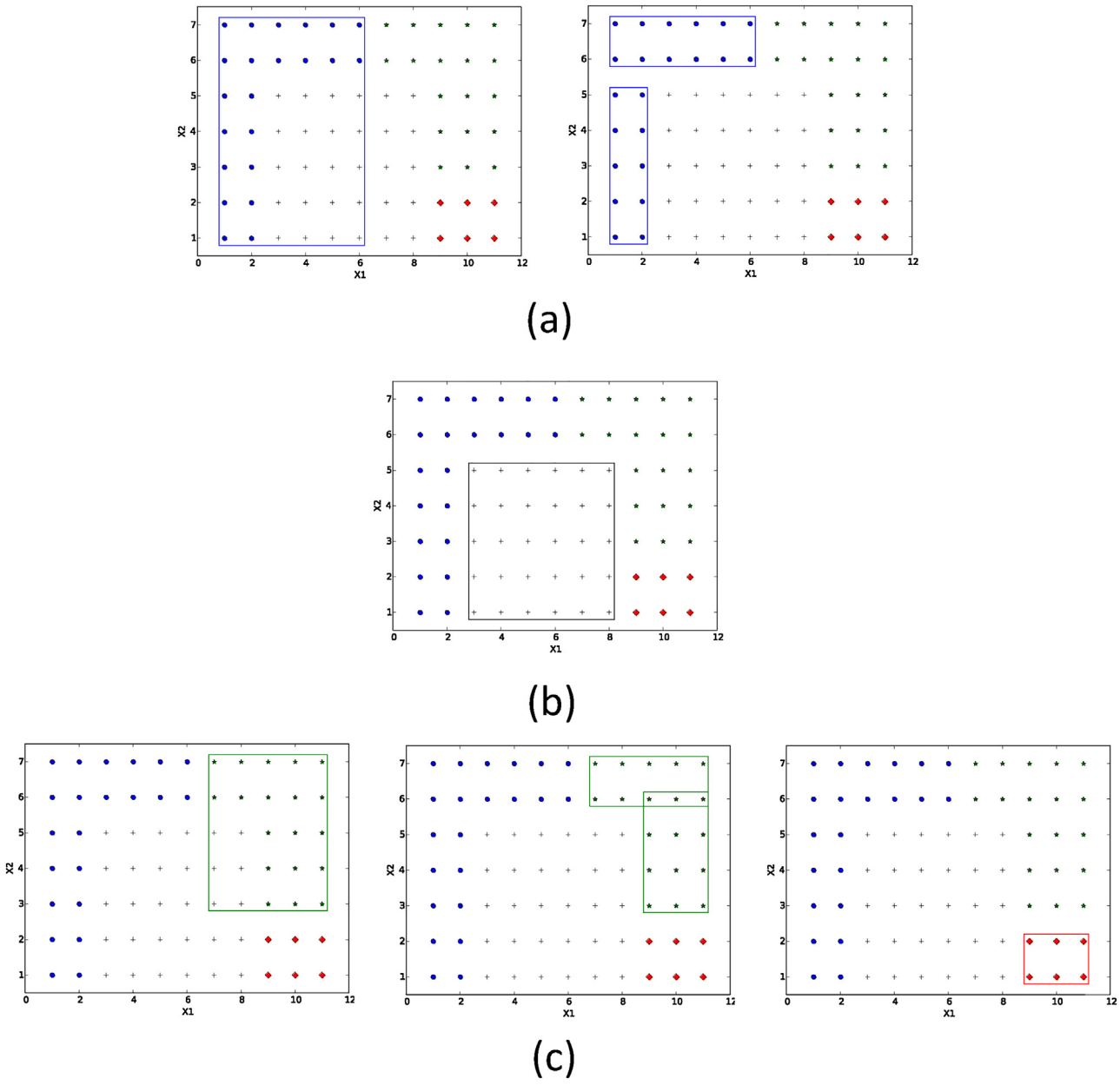

Fig. 7. Illustration for k-means++ initialization method. (a) Generated hyper-boxes with k-means++ initialization method for C^1. Left: When q is equal to 1, local error is equal to 20. Right: When q is equal to 2, local error is equal to zero; (b) hyper-box generated using the k-means++ initialization method for C^2. When q is equal to one, the local error is equal to zero; (c) Hyper-boxes generated using k-means++ initialization method for C^3 and for C^4. Left: When q is equal to one, local error is equal to six (for C^3). Middle: When q is equal to two, local error is equal to zero (for C^3). Right: When q is equal to one, local error is equal to zero (for C^4).

strates a learning performance which is superior to state-of-the-art alternatives for DMNN training algorithms, and a considerable reduction in dendrite number. Moreover, our training achieves performance similar to MLP and SVM.

For the DE algorithm, and for all the datasets, the population size was set as 10 individuals with 50 generations (these parameters were chosen based on preliminary experiments).

5.1. Results from synthetic datasets

In order to test the proposed training algorithm, it was first applied to two synthetic datasets with a high percentage of overlap, in which the overfitting problem becomes more apparent. It was compared with two algorithms, SLMP-P [24] and SLMP-R [8]; comparison focused mainly on SLMP-P, since this represents the state of the art for DMNN training algorithms.

The first was synthetic dataset A; this was composed of two classes and two features, and was generated using two Gaussian distributions with a standard deviation equal to 0.9. The first class was centered around point (0, 0) and the second class around point (1, 1). Fig. 9 (Left) shows dataset A.

Fig. 9 (Middle and Right) illustrates the generation and placement of the hyper-boxes in solving the classification problem A with the DE for DMNN and SLMP-P training algorithms respectively.

The second synthetic dataset was B; this was formed using three Gaussian distributions with a standard deviation equal to one. The classes were placed around points $(-1, -1)$, $(1, 1)$ and $(-1, 2)$. Fig. 10 (Left) exhibits the dataset B.

Fig. 10 (Middle and Right) shows the generations and the placement of the hyper-boxes for solving the classification problem B, using the DE for DMNN and SLMP-P training algorithms respectively.

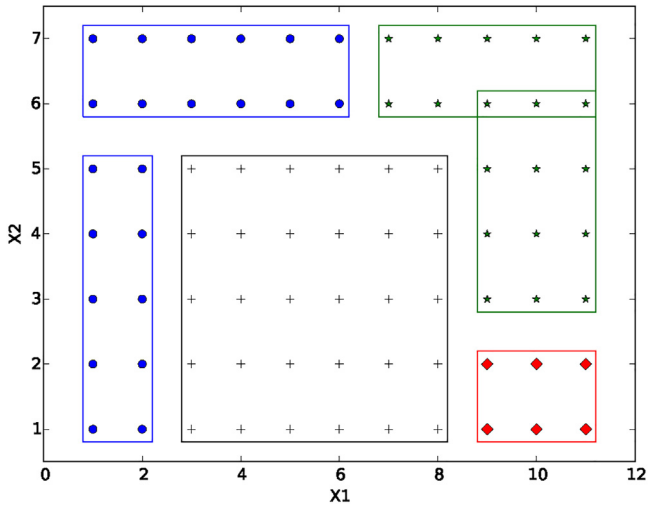

Fig. 8. Hyper-boxes generated using the k-means++ initialization method for this straightforward example.

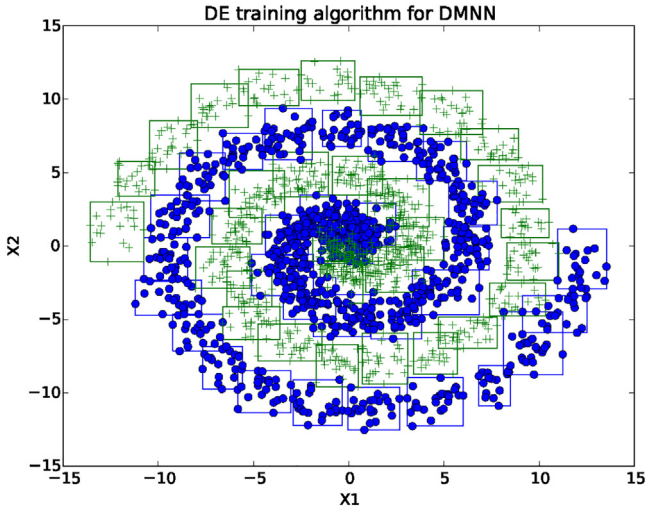

Fig. 11. Spiral 2. Spiral of two laps solved by DE for DMNN with 60 hyper-boxes.

The third synthetic dataset was Spiral 2, which is a double spiral with two laps. Fig. 11 displays Spiral 2 as solved by the proposed algorithm using the k-means++ initialization method. The final synthetic dataset was Spiral 10, which is similar to Spiral 2 except that it has ten spiral laps rather than two.

In these four synthetic datasets, 20% of the data was used for testing and 80% for training. Duplicate patterns were removed, the data was normalized, and all classifiers were tuned-up. Table 1 shows the hyper-parameters used, where *hL* is the number of hidden layers, *nu* is the number of neurons in the hidden layers, *LR* is the learning rate, *mo* is the momentum, *PD* is the polynomial degree and *nC* is the number of clusters. Table 2 presents the time in seconds taken during training for each type of classifier. In general, we see that the RBN based classifier is the fastest method; our method is the slowest. Nevertheless, the proposed method does not need an adjustment of the hyper-parameters unlike the other classifiers, which in most cases is time consuming.

Table 3 presents the classification errors and the number of dendrites obtained with the new training algorithm and with the actual best DMNN training algorithms (the SLMP-P and SLMP-R algorithms).

The proposed algorithm was compared also with some of the most common algorithms for classification: MLP, SVM and RBN. These classifiers were applied using the software WEKA 3.6 and 3.8 [32], while the DMNN training algorithms were programmed in Python 2.7. Table 4 describes the number features p of the datasets, the number of classes (n), the percentages used for training (P_{train})

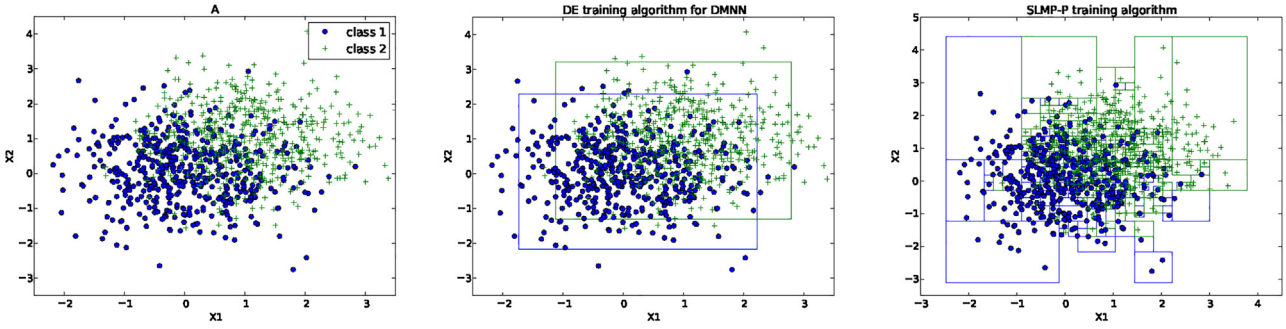

Fig. 9. Dataset A. Left: Distribution of dataset A composed of two classes and two features, generated by two Gaussian distributions with a standard deviation equal to 0.9. Middle: Two hyper-boxes generated with the DE training algorithm for DMNN. Right: 419 hyper-boxes generated with the SLMP-P training algorithm.

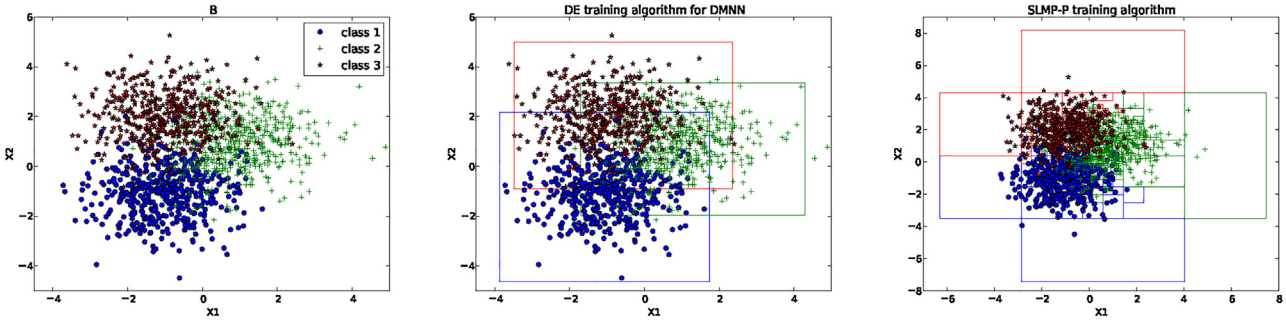

Fig. 10. Dataset B. Left: Distribution of dataset B composed of three classes and two features, generated by three Gaussian distributions with a standard deviation equal to one. Middle: Three hyper-boxes generated with the DE training algorithm for DMNN. Right: 505 hyper-boxes generated with the SLMP-P training algorithm.

Table 1
Hyper-parameters used for MLP, SVM and RBN.

	MLP				SVM	RBN
Dataset	*hL*	*nu*	*LR*	*mo*	*PD*	*nC*
A	1	2	0.3	0.2	3	3
B	1	10	0.3	0.2	2	4
Spiral 2	1	22	0.3	0.7	3	6
Spiral 10	1	40	0.9	0.1	5	10
Iris	1	5	0.3	0.2	1	2
Mammographic Mass	1	3	0.4	0.2	2	6
Liver Disorders	1	25	0.7	0.2	5	1
Glass Identification	2	2	0.2	0.1	1	4
Wine Quality	1	8	0.3	0.1	2	2
Mice Protein Expression	1	10	0.1	0.2	2	1
Abalone	1	17	0.1	0.2	2	10
Dermatology	1	20	0.3	0.2	3	2
Hepatitis	1	10	0.3	0.2	1	1
Pima Indians Diabetes	1	5	0.3	0.2	1	1
Ionosphere	1	18	0.3	0.2	3	5
Nao	1	5	0.3	0.2	4	6

Table 2
Classifier training times (in seconds).

Dataset	SLMP-R	SLMP-P	MLP	SVM	RBN	DE for DMNN
A	52.44	1.49	0.75	0.33	**0.12**	106.10
B	142.50	1.96	1.42	0.80	**0.10**	230.31
Spiral 2	53.21	1.00	3.18	3.84	**0.30**	5.18
Spiral 10	899.41	16.40	16.50	75.23	**0.16**	2213.33
Iris	0.60	0.20	0.14	0.20	**0.02**	33.25
Mammographic Mass	4.63	0.29	0.58	2.20	**0.20**	131.00
Liver Disorders	1.76	0.32	0.70	18.82	**0.10**	79.36
Glass Identification	0.39	0.60	0.68	4.00	**0.30**	210.05
Wine Quality	414.68	2.73	1.94	1.90	**0.45**	1700.14
Mice Protein Expression	8.93	1.66	5.67	**1.36**	3.77	10726.05
Abalone	3341.22	**16.21**	21.17	35.98	20.18	14242.10
Dermatology	2.18	**0.18**	1.75	3.00	0.19	1091.13
Hepatitis	0.04	0.02	0.12	0.50	**0.01**	46.58
Pima Indians Diabetes	10.53	0.69	0.40	1.00	**0.20**	197.62
Ionosphere	1.40	0.25	1.34	3.50	**0.20**	357.32
Nao	16.57	**0.14**	1.16	3.60	0.40	2.07

Bold values indicate least times achieved for each dataset.

Table 3
Comparison of SLMP-R, SLMP-P and DE for DMNN training algorithms for the four synthetic datasets.

	SLMP-R		SLMP-P			DE for DMNN		
Dataset	*K*	E_{test}	*K*	E_{train}	E_{test}	*K*	E_{train}	E_{test}
A	194	28.0	419	0.0	25.0	**2**	21.7	**20.5**
B	161	50.3	505	0.0	20.3	**3**	16.6	**15.2**
Spiral 2	160	8.6	356	0.0	7.2	**60**	7.3	**6.4**
Spiral 10	**200**	26.7	1648	0.0	10.6	1094	1.8	**6.3**

Bold values indicate best results achieved each specific dataset.

Table 4
Features of the datasets, percentages used for training and testing and best initialization methods.

Dataset	*p*	*n*	P_{train}	P_{test}	Method
A	2	2	80	20	HBd
B	3	2	80	20	HBd
Spiral 2	2	2	80	20	k-means++
Spiral 10	2	2	80	20	k-means++
Iris	3	4	75	25	HBd
Mammographic Mass	2	5	80	20	HBd
Liver Disorders	2	6	85	15	HBd
Glass Identification	6	10	75	25	HBd
Wine Quality	6	11	75	25	HBd
Mice Protein Expression	8	77	75	25	HBd
Abalone	28	8	80	20	HBd
Dermatology	6	34	80	20	HBd
Hepatitis	2	19	80	20	k-means++
Pima Indians Diabetes	2	8	80	20	HBd
Ionosphere	2	34	80	20	HBd
Nao	3	7	85	15	k-means++

Table 5
Comparison of SLMP-R, SLMP-P and DE for DMNN training algorithms for the 12 datasets.

	SLMP-R		SLMP-P			DE for DMNN		
Dataset	*K*	E_{test}	*K*	E_{train}	E_{test}	*K*	E_{train}	E_{test}
Iris	5	6.7	28	0.0	3.3	**3**	3.3	**0.0**
Mammographic Mass	51	14.4	26	0.0	19.2	**8**	15.8	**10.4**
Liver Disorders	41	42.0	183	0.0	35.5	**12**	37.6	**31.1**
Glass Identification	60	36.7	82	0.0	31.8	**12**	4.7	**13.6**
Wine Quality	120	51.0	841	0.0	42.1	**60**	42.1	**40.0**
Mice Protein Expression	77	18.9	809	0.0	5.0	**32**	6.6	**4.5**
Abalone	835	88.2	3026	0.0	80.6	**27**	77.1	**78.2**
Dermatology	192	57.8	222	0.0	15.5	**12**	4.8	**4.2**
Hepatitis	19	53.3	49	0.0	46.7	**9**	9.4	**33.3**
Pima Indians Diabetes	180	70.6	380	0.0	31.4	**2**	23.8	**23.5**
Ionosphere	238	10.0	203	0.0	35.7	**2**	2.8	**2.8**
Nao	231	13.3	74	0.0	**2.2**	**18**	3.7	7.8

Bold values indicate best results achieved each specific dataset.

Table 6
Comparison of MLP, SVM, RBN and DE for DMNN for all datasets.

	MLP		SVM		RBN		DE for DMNN	
Dataset	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}
A	20.7	24.0	20.8	22.0	21.8	23.5	21.7	**20.5**
B	15.5	16.7	15.7	16.7	15.5	17.0	16.6	**15.2**
Spiral 2	6.6	7.4	45.1	44.4	47.4	45.2	7.3	**6.4**
Spiral 10	48.8	48.9	49.5	49.6	49.4	45.6	1.8	**6.3**
Iris	1.7	**0.0**	4.2	**0.0**	4.2	**0.0**	3.3	**0.0**
Mammographic Mass	15.7	11.2	18.4	11.2	17.9	16.0	15.8	**10.4**
Liver Disorders	40.3	40.6	40.0	40.2	29.0	37.8	37.6	**31.1**
Glass Identification	14.1	20.4	12.3	18.2	0.0	20.4	4.7	**13.6**
Wine Quality	34.3	**39.0**	40.6	43.0	41.5	44.3	42.1	40.0
Mice Protein Expression	0.0	0.6	0.1	**0.5**	11.4	13.9	6.6	4.5
Abalone	75.0	**75.0**	73.1	**75.0**	72.0	76.0	77.1	78.2
Dermatology	0.0	**0.0**	1.4	1.4	1.0	2.8	4.8	4.2
Hepatitis	1.6	40.0	15.6	**33.3**	15.6	**33.3**	9.4	**33.3**
Pima Indians Diabetes	15.5	29.4	22.3	24.8	22.3	24.8	23.8	**23.5**
Ionosphere	0.3	7.1	6.8	6.8	6.4	8.6	2.8	**2.8**
Nao	4.5	**4.4**	31.8	32.2	31.5	31.1	3.7	7.8

Bold values indicate best results achieved each specific dataset.

and testing (P_{test}), and the initialization method (*Method*) which produced the best classification result.

For A, B, Spiral 2 and Spiral 10 datasets, DMNN trained with the proposed algorithm easily overcomes the SLMP-P and the SLMP-R obtaining the least classification errors and the least number of hyper-boxes. The SLMP-P presents a perfect classification performance in the training set, which leads to overfitting due to the increment in the model complexity. In contrast, our proposal avoids overfitting by choosing a good initial number of dendrites.

Comparing DMNN trained with the proposed algorithm with the rest of the classifiers (Table 6), DE based DMNN slightly overcomes MLP, SVM and RBN for A, B and Spiral 2. On the other hand, for Spiral 10 which is a more challenging dataset in machine learning, DE based DMNN easily places a set of hyper-boxes and obtains 6.3% error during testing. However, MLP obtains 48.9% error even though it uses 400 neurons in the hidden layer. This happens because neural networks are often better at responding to directions in space rather than to exact locations [33].

5.2. Results from real datasets

In this sub-section, the DE for DMNN was applied to 11 datasets from the UCI Machine Learning Repository [31].

The well-known Iris dataset was the first problem considered; this has four features (the length and the width of the sepal and petals in centimeters), three classes (Iris setosa, Iris virginica and Iris versicolor) and 50 samples per class.

The Mammographic Mass is a breast cancer screening dataset that has five attributes (BI-RADS assessment, age, shape, margin and density) and two classes (benign and malignant). The third dataset was Liver Disorders, which has two classes (the selector field used to split the data into two sets) and six features (five variables are blood tests, and one is the number of half-pint equivalents of alcoholic beverages drunk per day). This is a study which analyzes certain liver disorders which could arise from alcohol consumption.

Glass Identification is a classification study of glass types, which has 10 features (examples of chemical analysis) and six classes (six types of glasses). The Wine Quality dataset aims to predict the quality ranking from the chemical properties of wines, and consists of 11 features and six classes. The sixth dataset was Mice Protein Expression, which has 77 features (expression levels of 77 protein modifications measured in the cerebral cortex) and eight classes (eight classes of control and Down syndrome mice exposed to context fear conditioning, a task used to assess associative learning). The remaining datasets were Abalone, Dermatology, Hepatitis, Pima Indians Diabetes, Ionosphere and Nao.

The Abalone dataset is a collection of physical measurements of abalones, and consists of eight features (sex, length, diameter, height, whole weight, shucked weight, viscera weight and shell weight) and 28 classes (number of rings used to determine the age of the abalone). Dermatology was the eighth dataset, and this is a study of skin diseases consisting of 34 features (of which 12 are clinical attributes and 22 are histopathological attributes) and 6 classes (psoriasis, seborrheic dermatitis, lichen planus, pityriasis rosea, chronic dermatitis, and pityriasis rubra pilaris).

The hepatitis dataset provides information about patients affected by hepatic disease. The dataset has a mixture of 19 real and integer-valued features and two classes (histology).

The next dataset was Pima Indians Diabetes, which is research data on female patients of at least 21 years old of Pima Indian heritage. This dataset contains eight features with clinical information related to diabetes and two classes (tested positive or negative for diabetes).

The last dataset was Ionosphere from the UCI Machine Learning Repository, which has 34 continuous features (collected from a phased array of sixteen high-frequency antennas) and two classes; the data involves either good (radar returns showing structure in the ionosphere) or bad returns.

Table 5 presents the results of DMNN training algorithms for the UCI datasets. In most cases, the proposed algorithm achieves the smallest classification error, and the number of dendrites is much lower than in the other methods. This allows implementation of this classifier in embedded devices to obtain real-time responses.

In analyzing the results of the actual best training algorithm for DMNN (the SLMP-P [24]) with the proposed algorithm, it is observed that the SLMP-P algorithm has the problem of overfitting for these datasets too. This is because SLMP-P fits very well during training, but fails during generalization when testing, since this technique does not have any method of regularization. For this reason, DE for DMNN outperforms SLMP-P for the synthetic and real datasets.

Table 6 shows a comparison between DE based DMNN and the rest of the classifiers, and demonstrates that the DE training algorithm for DMNN often achieves the lowest classification error for the real datasets with low dimensions. Nevertheless, when working with a high dimensional dataset (Abalone, Mice Protein Expression and Wine Quality) the other classifiers defeat our proposal for two reasons: (1) the DMNN architecture does not have a feature extraction layer, and (2) training time for differential evolution can be very time consuming to look for the best solution.

As a comparison of the performance of the proposed algorithm in statistical terms with the previously mentioned classifiers (with a significance level $\alpha = 0.05$), Table 7 gives the p-values obtained in a paired t-test with the other methods. For SLMP-R, SLMP-P and RBN the p-values are less than the α value so the null-hypothesis is rejected. On the other hand, for MLP and SVM the p-values are greater than the α value so it is not rejected. The null-hypothesis means that our method and the other classifiers provide a similar accuracy. Based on these results, we can say that DE for DMNN has a similar accuracy to MLP and SVM and a better performance than RBN, SLMP-R and SLMP-P for these datasets. Furthermore, DE for DMNN achieved an average accuracy of 81%, MLP 77% and SVM 73% for all used datasets.

Table 7
We show p-values of a paired t-test with $\alpha = 0.05$ in order to statistically evaluate the accuracy of the besought algorithm with respect to the other classifiers.

Classifiers	*p-Values*
SLMP-R	0.0002
SLMP-P	0.0002
MLP	0.0760
SVM	0.0269
RBN	0.0082

Table 8
Results of applying 10-fold cross-validation to all the datasets with DE for DMNN.

Dataset	*best*	*mean*	*std*
A	15.8	19.2	2.0
B	12.8	15.6	2.3
Spiral 2	3.6	7.7	2.5
Spiral 10	3.6	7.7	2.4
Iris	0.0	1.3	2.7
Mammographic Mass	0.0	3.0	6.2
Liver Disorders	8.8	27.5	9.9
Glass Identification	0.0	6.4	6.5
Wine Quality	42.3	44.8	1.5
Mice Protein Expression	2.8	5.4	1.1
Abalone	75.8	79.3	1.8
Dermatology	0.0	5.1	3.3
Hepatitis	14.3	30.0	10.0
Pima Indians Diabetes	15.8	26.0	4.7
Ionosphere	2.8	6.0	4.0
Nao	2.9	6.8	3.4

Finally, Table 8 displays the results of applying 10-fold cross-validation to all the datasets with DE for DMNN, where "*best*" indicates the best result, "*mean*" is the mean and "*std*" is the standard deviation of the folds for each dataset. DE for DMNN has a relatively small standard deviation for most of the datasets (except for Mammographic, Liver Disorders, Glass Identification, Hepatitis and Pima Indians Diabetes), which demonstrates the robustness of the method.

5.3. Figure recognition with Nao robot

In order to test the performance of the new algorithm in resolving a real practical problem, the proposed training algorithm was used to classify geometric figures (a rectangle, a circle and a pentagon) using a Nao robot. For this experiment, 600 pictures were taken of the three figures in different positions and orientations.

In the pre-processing stage, all the pictures were transformed from the RGB (Red/Green/Blue) format to the CIELUV (Luminance, Saturation, and Hue angle) or HSV (Hue, Saturation and Value) color spaces, depending on the grade of illumination. Following this, the images were binarized and seven invariant Hu moments of the processed images were obtained.

Finally, the classifier was trained and tested on the robot, and obtained a 98% classification score in practice. Fig. 12 shows the Nao robot placed to classify the geometric figures.

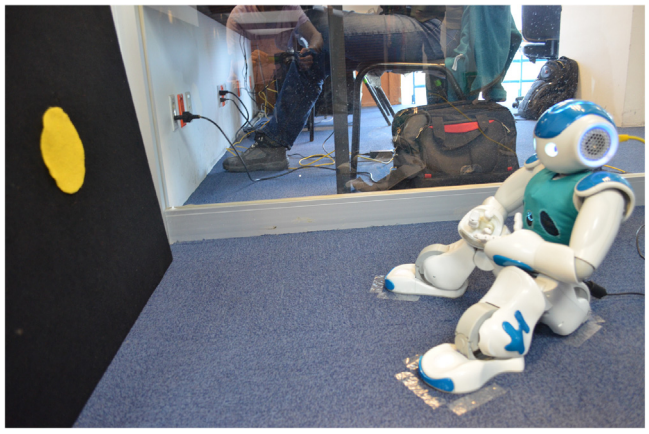

Fig. 12. Nao placed to classify the geometric figures.

This experiment was developed to aid a laboratory study which focuses on teaching geometric figures and colors to children from two to three years old using a robot.

6. Conclusions and future work

This work proposes a method for evolving the dendrites (hyper-boxes) of a morphological neuron to optimize the classification error. Unlike this proposed approach, most training techniques in literature for morphological neural networks are based on intuition in terms of where to place the hyper-boxes without adding an optimization step to increase their effectiveness as a classifier. Two methods are proposed for generating the initial population of hyper-boxes: one based on the division of an initial hyper-box and the other on an initial clustering using k-means++. Both methods are useful, depending on the specific classification problem in reducing the number of evolutionary generations.

The proposed method performance is evaluated experimentally using four synthetic databases and 11 real databases as benchmarks. When comparing the DE for DMNN with state-of-the-art methods in morphological neural networks, the results show that our method achieves fewer misclassifications and requires fewer dendrites. Compared to most popular techniques in this area of machine learning, the results show that DE for DMNN outperforms RBN and presents a competitive performance compared with the MLP and SVM classifiers (especially in datasets with low dimensionality).

Furthermore, we invite the reader to consider that a performance equivalent to that of MLP and SVM was achieved without using feature extraction layers (with only one dendrite layer). Finally, the implementation of these dendrite morphological neurons was shown to be satisfactory in a Nao robot in an application involving geometric figures recognition. It should be clarified that the greatest disadvantage of our method is the training time required due to evolutionary optimization. For this reason, future work will involve the implementation of the proposed algorithm on a Graphic Processing Unit (GPU) focusing on applications requiring real-time responses, such as object classification and segmentation.

Acknowledgements

E. Zamora and H. Sossa would like to acknowledge UPIITA-IPN and CIC-IPN for the support to carry out this research. This work was economically supported by SIP-IPN (grant numbers 20160945, 20170836, 20180180 and 20161116, 20170693, 20180730), and CONACYT (grant number 155014 (Basic Research) and grant number 65 (Frontiers of Science)). R. Barrón is grateful for the SIP-IPN (grant numbers 20160828) support. And F. Arce acknowledges CONACYT for the scholarship granted towards pursuing his PhD studies.

References

[1] F. Arce, E. Zamora, H. Sossa, R. Barrón, Dendrite morphological neural networks trained by differential evolution, 2016 IEEE Symposium Series on Computational Intelligence (SSCI) (2016) 1–8, http://dx.doi.org/10.1109/SSCI.2016.7850259.
[2] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359, http://dx.doi.org/10.1023/A:1008202821328.
[3] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution – A Practical Approach to Global Optimization, Natural Computing, Springer-Verlag, 2005, ISBN:540209506.
[4] V. Feoktistov, Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[5] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, MIT Press, Cambridge, MA, USA, 1986, pp. 318–362, Ch. Learning Internal Representations by Error Propagation.
[6] D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Syst. 2 (1988) 321–355.
[7] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297, http://dx.doi.org/10.1023/A:1022627411411.
[8] G.X. Ritter, L. Iancu, G. Urcid, Morphological perceptrons with dendritic structure, The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ '03, vol. 2 (2003) 1296–1301, http://dx.doi.org/10.1109/FUZZ.2003.1206618.
[9] G.X. Ritter, M.S. Schmalz, Learning in lattice neural networks that employ dendritic computing, IEEE International Conference on Fuzzy Systems, 2006 (2006) 7–13, http://dx.doi.org/10.1109/FUZZY.2006.1681687.
[10] G.X. Ritter, D. Li, J.N. Wilson, Image Algebra and Its Relationship to Neural Networks, 1989, http://dx.doi.org/10.1117/12.960428.
[11] J.L. Davidson, G.X. Ritter, Theory of Morphological Neural Networks, 1990, http://dx.doi.org/10.1117/12.18085.
[12] J.L. Davidson, K. Sun, Template Learning in Morphological Neural Nets, 1991, http://dx.doi.org/10.1117/12.46114.
[13] J.L. Davidson, F. Hummer, Morphology neural networks: an introduction with applications, Circ. Syst. Signal Process. 12 (2) (1993) 177–210, http://dx.doi.org/10.1007/BF01189873.
[14] G.X. Ritter, P. Sussner, An introduction to morphological neural networks, Proceedings of the 13th International Conference on Pattern Recognition, 1996, vol. 4 (1996) 709–717, http://dx.doi.org/10.1109/ICPR.1996.547657.
[15] G.X. Ritter, G. Urcid, Lattice algebra approach to single-neuron computation, IEEE Trans. Neural Netw. 14 (2) (2003) 282–295, http://dx.doi.org/10.1109/TNN.2003.809427.
[16] P. Sussner, Morphological perceptron learning, Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings (1998) 477–482, http://dx.doi.org/10.1109/ISIC.1998.713708.
[17] L.F. Pessoa, P. Maragos, Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition, Pattern Recogn. 33 (6) (2000) 945–960, http://dx.doi.org/10.1016/S0031-3203(99)00157-0 http://www.sciencedirect.com/science/article/pii/S0031320399001570.
[18] A. Barmpoutis, G.X. Ritter, Orthonormal basis lattice neural networks, 2006 IEEE International Conference on Fuzzy Systems (2006) 331–336, http://dx.doi.org/10.1109/FUZZY.2006.1681733.
[19] G.X. Ritter, G. Urcid, Learning in Lattice Neural Networks that Employ Dendritic Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 25–44, http://dx.doi.org/10.1007/978-3-540-72687-6_2.
[20] P. Sussner, E.L. Esmi, Constructive Morphological Neural Networks: Some Theoretical Aspects and Experimental Results in Classification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 123–144, http://dx.doi.org/10.1007/978-3-642-04512-7_7.
[21] P. Sussner, E.L. Esmi, An introduction to morphological perceptrons with competitive learning, 2009 International Joint Conference on Neural Networks (2009) 3024–3031, http://dx.doi.org/10.1109/IJCNN.2009.5178860.
[22] P. Sussner, E.L. Esmi, Morphological perceptrons with competitive learning: lattice-theoretical framework and constructive learning algorithm, Inf. Sci. 181 (10) (2011) 1929–1950, http://dx.doi.org/10.1016/j.ins.2010.03.016, special Issue on Information Engineering Applications Based on Lattices, http://www.sciencedirect.com/science/article/pii/S0020025510001283.
[23] R. de, A. Araujo, A morphological perceptron with gradient-based learning for Brazilian stock market forecasting, Neural Netw. 28 (2012) 61–81, http://dx.doi.org/10.1016/j.neunet.2011.12.004 http://www.sciencedirect.com/science/article/pii/S0893608011003200.
[24] H. Sossa, E. Guevara, Efficient training for dendrite morphological neural networks, Neurocomputing 131 (2014) 132–142, http://dx.doi.org/10.1016/j.

neucom.2013.10.031 http://www.sciencedirect.com/science/article/pii/S0925231213010916.
[25] G.X. Ritter, G. Urcid, V.N. Juan-Carlos, Two lattice metrics dendritic computing for pattern recognition, 2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (2014) 45–52, http://dx.doi.org/10.1109/FUZZ-IEEE.2014.6891551.
[26] H. Sossa, E. Guevara, Modified Dendrite Morphological Neural Network Applied to 3D Object Recognition, LNCS 7914, Springer-Verlag, 2013, pp. 314–324, http://dx.doi.org/10.1007/978-3-642-38989-4_32.
[27] H. Sossa, G. Cortés, E. Guevara, New Radial Basis Function Neural Network Architecture for Pattern Classification: First Results, Springer International Publishing, Cham, 2014, pp. 706–713, http://dx.doi.org/10.1007/978-3-319-12568-8_86.
[28] E. Zamora, H. Sossa, Dendrite morphological neurons trained by stochastic gradient descent, 2016 IEEE Symposium Series on Computational Intelligence (SSCI) (2016) 1–8, http://dx.doi.org/10.1109/SSCI.2016.7849933.
[29] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.
[30] D. Ardia, K. Boudt, P. Carl, K.M. Mullen, B.G. Peterson, Differential evolution with DEoptim: An application to non-convex portfolio optimization, R Journal 3 (1) (2011) 27–34.
[31] D.N.A. Asuncion, UCI Machine Learning Repository, 2007.
[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The Weka Data Mining Software: An Update, 2009.
[33] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016 http://www.deeplearningbook.org.

Fernando Arce is a full-time PhD student in Computer Science at the Centre for Computing Research of the Instituto Politécnico Nacional. He received a B.Sc. Degree in Electronics from the Instituto Tecnológico Superior de Cajeme (2010) and a Master's Degree in electrical engineering from CINVESTAV-IPN (2014), Campus Zacatenco. His main research interests are Computer Vision and Neural Networks.

Erik Zamora is a full professor with the National Polytechnic Institute of Mexico. He received his Diploma in Electronics from UV (2004), his Master's degree in Electrical Engineering from CINVESTAV (2007), and his PhD in automatic control from CINVESTAV (2015). He developed the first commercial Mexican myoelectric system to control a prosthesis in the Pro/Bionics company and a robotic navigation system for unknown environments guided by emergency signals at the University of Bristol. He has a postdoctoral position in CIC-IPN. His current interests include autonomous robots and machine learning. He has published eight technical papers in international conference proceedings and journals and has directed seven theses on these topics.

Humberto Sossa was born in Guadalajara, Jalisco, Mexico in 1956. He received his B.Sc. degree in Electronics from the University of Guadalajara in 1981, his M.Sc. in Electrical Engineering from CINVESTAV-IPN in 1987, and his Ph.D. in Informatics from the National Polytechnic Institute of Grenoble, France in 1992. He is a full-time professor at the Centre for Computing Research of the National Polytechnic Institute of Mexico. His main research interests are in pattern recognition, artificial neural networks, image analysis, and robot control using image analysis.

Ricardo Barrón holds a PhD in Computer Science from CIC-IPN and a Master's degree in CE from CIC-IPN. He is a full time researcher at the Artificial Intelligence Laboratory of CIC-IPN with research interests including pattern recognition, GPU computing, evolutionary computation and parallel systems.