Dendrite Morphological Neural Networks Trained by Differential Evolution

Fernando Arce\textsuperscript{1}, Erik Zamora\textsuperscript{2}, Humberto Sossa\textsuperscript{1}, Ricardo Barrón\textsuperscript{1}
Instituto Politécnico Nacional
CIC\textsuperscript{1}, UPIITA\textsuperscript{2}
Av. Luis Enrique Erro S/N, Unidad Profesional Adolfo López Mateos, C.P. 07738
Mexico City
Emails: b150689@sagitario.cic.ipn.mx, ezamora@ipn.mx, hsossa@cic.ipn.mx, rbarron@cic.ipn.mx

Abstract—A new efficient training algorithm for a Dendrite Morphological Neural Network is proposed. Based on Differential Evolution, the method optimizes the number of dendrites and increases classification performance. This technique has two initialisation ways of learning parameters. The first selects all the patterns and opens a hyper-box per class with a length such that all the patterns of each class remain inside. The second generates clusters for each class by k-means++. After the initialisation, the algorithm divides each hyper-box and applies Differential Evolution to the resultant hyper-boxes to place them in the best position and the best size. Finally, the method selects the set of hyper-boxes that produced the least error from the least number. The new training method was tested with three synthetic and six real databases showing superiority over the state-of-the-art for Dendrite Morphological Neural Network training algorithms and a similar performance as well as a Multilayer Perceptron, a Support Vector Machine and a Radial Basis Network.

I. INTRODUCTION

Linear perceptrons divide the input space into several regions, using hyper-planes as decision boundaries. In a case where more linear layers are connected, perceptrons divide the input space by a hyper-surface. In contrast, Morphological Neural Networks (MNN) divide the same space by several piecewise lines that together can create complex decision boundaries, allowing non-linear classification with only one neuron. This is not viable with a one-layer linear perceptron. The morphological processing involves min and max operations. These operators generate the piecewise boundaries for classification problems, and have the advantage of being implemented easily in logic devices.

This paper focuses on a specific type of Morphological Network which is called a Dendrite Morphological Neural Network (DMNN). A neuron has dendrites, and each represents a hyper-box in high dimensional space (and a box in 2D). These dendrites have played an important role in the proposal of training methods. The common approach is to enclose patterns with hyper-boxes and label each to the correct class. There are many training methods that have been published based on this approach. Most are heuristics and are not based on the optimization of learning parameters. In this paper, we show that a parameter optimization can be computed and can improve the classification performance of DMNN over heuristic-based and other popular machine learning methods.

Differential Evolution is distinguished as a powerful optimization technique [1], [2], [3]. This method is especially useful for problems with cost functions that are non-differentiable, non-continuous, non-linear, multi-dimensional or have many local minima. The main advantage is that the method does not require calculating analytic expressions such as gradients. Furthermore, the method is robust to find the global optimization in comparison with gradient-based methods. In this paper, we adapt the Differential Evolution to find the number of dendrites and their dendrite parameters for classification problems. This is useful because a DMNN has a cost function that is non-continuous due to the argmax operator at the neuron output.

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; 2) our proposal outperforms the most effective heuristic-based and other popular machine learning methods; 3) the code for our approach is available at [4].

The following sections are organized as follows. Section 2 provides the previous work of MNN. Section 3 introduces the DMNN computational basis. Section 4 presents the proposed algorithm. Section 5 discusses the experimental results to assess the effectiveness of our proposal. Finally, in Section 6 we give our conclusions and proposed future work.

II. PREVIOUS WORK

Morphological neurons were proposed by Davidson and Ritter in their seminal paper [5]; their computing capabilities were studied by Ritter and Sussner [6], [7]. Taking into account that information processing occurs also in dendrites, not only in cell body of neurons [8], the dendrite morphological neurons were proposed as an extension in [9]. A key issue is their training; we need to determine automatically the number of dendrites and the dendrite weight values. Several training approaches have been proposed [9], [10], [11], [12], [13], [14], [15], [16], [17]. And some real problems have been solved by using MNN [18] [19] [20] [21] [22].

Most of these methods use heuristics to determine the location of the hyper-boxes in the input space. In contrast, we propose making a search of those that were best classified based on an evolutionary approach. This is not the first time that parameters optimization has been proposed in the

The main advantage of optimization based on evolution is that we can do without analytically evaluating expressions, e.g. gradients, and also deal with discontinuities of the cost function.

III. COMPUTATIONAL BASIS FOR DENDRITE DMNN

MNN base their computations on Lattice algebra. More detailed information can be found in [25], [26].

For this research paper, we used a total connected DMNN with a single output neuron. The computation performed $\tau_k^j$ by $k$th dendrite for the $j$th class can be expressed by (1):

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

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

On the other hand, the output value of a DMNN with a single output neuron $\tau_j$ is the maximum argument calculated with (2):

$$\tau_j = \arg\max_k(\tau_k^j),$$

where the $\arg\max_k$ function selects only one dendrite from all dendrites of the network, $\tau_k^j$ vector.

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.

IV. THE PROPOSED TRAINING ALGORITHM

This section is dedicated to explain the new training algorithm steps. Before presentation, two definitions are provided:

Definition 1. Let $x_1, x_2, \ldots, x_m$ be a finite set of sample patterns, where each pattern $x_j = (x_{j1}, x_{j2}, \ldots, x_{jn}) \in \mathbb{R}^n$ for $i = 1, 2, \ldots, m$ is a $n$-dimensional vector. Furthermore, each pattern $x_j$ 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 a $n$ dimensional box that contains a finite set of patterns $x \in \mathbb{R}^n$. This $HB^n$ defines the weights $w_{ik}^l$, $l = \{0, 1\}$ where 1 denotes excitation and 0 inhibition.

Equation (3) defines a hyper-box:

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

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

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 parents:
            Parent is replaced by the offspring.
        Until a stop condition is satisfied.
End
```

A. Training Algorithm

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

- **HBd initialisation:**
  1) For each $C^j$ class, select all the patterns and open a hyper-box per class, with a length such that all the patterns of each class remain inside. Sussner and Laureano [14] proposed a similar algorithm which keeps dividing them in order to avoid overlapping. The suggested algorithm does not care about the hyper-boxes overlapping.
  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.

- **K-means++ initialisation:**
  1) Generates $Nq$ clusters for each $C^j$ class by using the clustering algorithm k-means++ [27]. These clusters are transformed to hyper-boxes.
  2) Apply DE as [28] to the actual hyper-boxes.
  3) Select the best set of hyper-boxes that produced the least error. Algorithm 1 shows the pseudo-code of DE applied to a DMNN.

a) **Initialization:** The set of hyper-boxes conforms a $k \times n$ matrix called ListH. Each row from ListH has 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 an 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 by (4) dividing the $g$ number of misclassified patterns by the total number of the $m$ patterns:

$$\%error = \frac{g}{m} \times 100,$$  

(4)
\[ \tau_{ij}(x) > 0 \]
\[ \tau_{ij}(x) = 0 \]
\[ \tau_{ij}(x) < 0 \]

\[ \text{List}_{H_t} = \text{List}_{H_a} + F \times (\text{List}_{H_b} - \text{List}_{H_c}), \quad (5) \]

where \( a, b, c \) must be distinct. \( \text{List}_{H_t} \) is the offspring, \( F \) is a random vector, \( \text{List}_{H_b} \) and \( \text{List}_{H_c} \) are randomly chosen parents and \( \text{List}_{H_a} \) is the best member of the population.

f) \textit{Stop condition}: DE stops if satisfying the error criteria or the \( q \) number of iterations.

B. Example of the Proposed Training Algorithm

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

In order to explain this training algorithm, a straightforward example of three classes with two features is utilized, with the HBd initialisation method. Fig. 2 presents the problem to solve, blue dots belong to \( C_1 \), green dots belong to \( C_2 \) and red dots to \( C_3 \).

1) Select all the patterns of each class and open a hyper-box per class. Fig. 3 shows the hyper-boxes that covers 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. 4 exhibits the divisions of each hyper-box generated in step 2 when \( d = 1, d = 2 \) and \( d = 3 \). When \( d \) is equal to 1 there is no division.

3) Select the best set of hyper-boxes. Fig. 5 displays the best generated after applying the training algorithm, with \( d = 3 \).

V. EXPERIMENTAL RESULTS

Validation experimental results are presented in this section. The experiments were performed with three synthetic datasets and with six real datasets from the UCI Machine Learning...
Repository [29]. On the basis of these results, this new training algorithm demonstrates a superior learning performance over the state-of-the-art for DMNN training algorithms and a considerable reduction of the dendrite number. Moreover, our training achieves similar performances to the most common classifiers.

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

### A. Experimental Results Using Synthetic Data

In order to test the proposed training algorithm, it was first applied to two synthetic datasets with a high percent of overlapping where the overfitting problem can be more appreciable. It was compared with the algorithms SLMP-P [17] and SLMP-R [25] but mainly with the SLMP-P because it actually represents the state-of-the-art for DMNN training algorithms.

The first synthetic dataset was “A”, composed of two classes and two features, generated by two Gaussian distributions with a standard deviation equal to 0.9. The first class was centered around the point (0,0) and the second class around the point (1,1). Fig. 6 shows the dataset “A”.

Fig. 7 presents the generation and placement of the hyper-boxes to solve the classification problem “A” with the DE for DMNN and SLMP-P training algorithms.

“B” was the second synthetic dataset and was formed by three Gaussian distributions with a standard deviation equal to 1.0. The classes were placed around the points (-1,-1), (1,1) and (-1,2). Fig. 8 exhibits the dataset “B”.

Fig. 9 shows the generations and the placement of the hyper-boxes to solve the classification problem “B” with the DE for DMNN and SLMP-P training algorithms.

The third synthetic dataset was “Spiral 2” which is a double spiral with two laps. Fig. 10 displays “Spiral 2” solved by the proposed algorithm; this dataset was the only one which used the K-means++ initialisation.

In these three synthetic datasets, 20% of data was used for testing and 80% for training. Table I 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 reduc-
Fig. 7. Hyper-boxes generated by the DE for DMNN and SLMP-P training algorithms for problem “A”. Left: 2 hyper-boxes generated with DE training algorithm for DMNN. Right: 419 hyper-boxes generated with SLMP-P training algorithm.

Fig. 8. Dataset “B” composed of three classes and two features, generated by three Gaussian distributions with a standard deviation equal to 1.0.

B. Experimental Results Using Real Data

In this sub-section, the DE for DMNN was applied to real data (with \( p \) classes and \( n \) features) from the UCI Machine Learning Repository [29].

For the three synthetic and for the real datasets, the proposed algorithm was compared with the most common algorithms for classification: MLP [30], SVM [31] and RBN [32]. These final algorithms were applied using the software WEKA 3.6 and 3.8 [33] and the DMNN training algorithms were programed in Python 2.7.

In all the datasets, duplicated patterns were removed and the data was normalized; and all the classifiers were tuned-up. Table II 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 III presents the results with DMNN training algorithms for the six UCI datasets. In all cases, the proposed algorithm achieves the smallest classification error and the number of dendrites is much less than the others. This allows us to implement this classifier in embedded devices to obtain real time responses.

Analyzing the results of the actual best training algorithm for DMNN (the SLMP-P [17]) with the proposed algorithm, it can be appreciated that the SLMP-P has the problem of overfitting. This is because the SLMP-P fits very well in the training set but fails to generalize in the testing set due to this technique being a heuristic and not an optimization method.

For this reason, DE for DMNN overcomes the SLMP-P.

Table IV makes a comparison between the DE for DMNN
and the rest of the classifiers and shows that the DE training algorithm for DMNN not always achieved the least classification error, but was one of the best classifiers with the actual datasets.

In order to compare statistically the performance of the proposed algorithm with the previously mentioned classifiers (with a significance level $\alpha = 0.05$), Table V exhibits the obtained p-values in a paired t-test with the other methods.

Based on these results, we can say that the DE for DMNN has a similar performance to the MLP, the SVM and the RBN, for the used datasets.

Finally, Table VI displays the results of applying 10-fold cross-validation to the nine datasets with DE for DMNN; where best is the best result, mean is the mean and std is the standard deviation of the folds for each dataset. DE for DMNN has a small standard deviation for most of the datasets (except for Mammographic, Liver Disorders and Glass Identification) which shows the method robustness.

VI. CONCLUSIONS AND FUTURE WORK

The research presented in this paper is an efficient and novel training algorithm for DMNN. Comparisons of the sought algorithm with the state-of-the-art for DMNN training algorithms demonstrated that DE for DMNN achieved the
require real time responses such as object classification and segmentation.

**ACKNOWLEDGMENT**

E. Zamora and H. Sossa would like to acknowledge the support provided by UPIITA-IPN and CIC-IPN to carry out this research. This work was economically supported by SIP-IPN (grants numbers 20160945 and 20161116), and CONACYT [grant number 155014 (Basic Research) and grant number 65 (Frontiers of Science)]. F. Arce acknowledges CONACYT for the scholarship granted to pursue his Ph.D. studies.

**REFERENCES**


