Generalized Neural Trees for Pattern Classification

Gian Luca Foresti and Christian Micheloni

***Abstract*—In this paper, a new neural tree (NT) model, the generalized NT (GNT), is presented. The main novelty of the GNT consists in the definition of a new training rule that performs an overall optimization of the tree. Each time the tree is increased by a new level, the whole tree is reevaluated. The training rule uses a weight correction strategy that takes into account the entire tree structure, and it applies a normalization procedure to the activation values of each node such that these values can be interpreted as a probability. The weight connection updating is calculated by minimizing a cost function, which represents a measure of the overall probability of correct classification. Significant results on both synthetic and real data have been obtained by comparing the classification performances among multilayer perceptrons (MLPs), NTs, and GNTs. In particular, the GNT model displays good classification performances for training sets having complex distributions. Moreover, its particular structure provides an easily probabilistic interpretation of the pattern classification task and allows growing small neural trees with good generalization properties.**

***Index Terms*—Neural networks (NNs), neural trees (NTs), pattern classification, performance evaluation, search methods.**

I. Introduction

NEURAL TREES (NTs) were introduced for pattern classification in an attempt to combine the advantages of decision trees (DTs) [1] and neural networks (NNs) [2], so that they could have a fast training phase, as in DTs, and a strong performance in classification. In 1988, Utgoff proposed a hybrid NN (called *perceptron tree*) with a tree-based structure that was similar to DTs, but having perceptrons as leaves [3]. In 1990, Sirat and Nadal [4] published (almost at the same time as Marchand and Golea [5]) a paper concerning binary NTs, in which perceptrons are used at each node of the tree. Some years after, Sankar and Mammone [6]–[8] proposed an NT network for multiclass problems that consists of single-layer NNs connected to a tree structure and uses a learning procedure based on the $\mathcal{L}1$ norm of the classification error (measured as the distance between generated outputs and targets). For n-class problems, they use n neurons and a local encoding scheme that classifies an input pattern according to a *winner-take-all* rule. Each node divides the training data into n convex regions: thus, there are at most n children for each node. Rahim [9] and Farrell *et al.* [10] used variations of this model for different recognition problems.

The standard NT model consists of multiple NNs connected into a tree architecture, where each NN is used to recursively partition the feature space into regions and uses (for its training) only those patterns (local training set) that fall into this region. The basic idea of this approach is that learning the training set by dividing it into local sets (according to the subregions into which the feature space is divided) is easier than learning it whole. DTs also adopt such a technique, but often they use hyperplanes that are perpendicular to the axes of the feature space. NTs have no restrictions regarding the orientation of hyperplanes, so they are more adapted to capturing the pattern distribution. Sethi [11] proposed a procedure for transforming a DT into a multilayer feedforward NN, called *entropy network*. This methodology, which uses a rule for incremental learning, specifies the number of neurons needed in each layer and allows each layer to be trained separately. Actually, some DT learning algorithms (e.g., CART [12]) allow the use of nonperpendicular hyperplanes, but they are computationally expensive due to the large number of hyperplanes that must be evaluated. Several methods exist which determine (in parts) the architecture of the NN as well: pruning [7], [8], cascade correlation [13], dynamic learning vector quantization [15], and support vector machines [16], etc.

In recent years, some NN models with a tree topology have been developed for particular applications, or to cope with difficulties such as large pattern sets or complex pattern distributions in high-dimensional feature spaces. Li *et al.* [16] proposed a structured parameter adaptive (SPA) NT which is a multilevel competitive NN with a tree topology. In this network, Hebbian learning algorithm is used for the adaptation of the network parameters, and some specific operations are used for adapting the tree structure: the creation of new nodes, the splitting of nodes into more nodes, and the deletion of nodes from the network. Hebbian learning is employed both during training and execution to achieve parameter adaptation, while other methods employ combinatorial or heuristic search algorithms during training and no changes during execution. Behnke and Karayiannis [17] proposed a competitive NT (CNeT) for pattern classification that combines the advantages of competitive NNs and DTs. The CNeT contains m-ary nodes, grows during the learning phase by using inheritance to initialize new nodes, and employs competitive learning. Forward pruning can control the growth of the tree. Song *et al.* [18] proposed a structural adaptive intelligent tree (SAINT). The input feature space is hierarchically partitioned by using a tree-structured network that preserves a lattice topology at each subnetwork. Experimental results reveal that SAINT is very effective for the classification of a large set of real-world handwritten characters with high variations, as well as multilingual, multifont, and multisize large-set characters. The SAINT model is closely related to the SPA model in that it hierarchically partitions the n-dimensional feature space into successive subregions. However, SAINT differs from SPA in that each of the subnetworks has a different topology. They achieve a better classification performance than MLPs in problems characterized by large sets of classes but

Manuscript received October 28, 1999; revised November 27, 2000 and May 16, 2002.

The authors are with the Department of Mathematics and Computer Science (DIMI), University of Udine, 33100 Udine, Italy.

Digital Object Identifier 10.1109/TNN.2002.804290

worse classification with a medium or small number of classes. Song *et al.* showed that SAINT has a worse classification rate than MLPs when the number of classes is less than 50.

In this paper, a new NT model, called generalized NT (GNT), is presented. The main novelty of this model is that it does not train each node using the pattern contained in its subregion (i.e., the local training set), but instead trains the whole tree using a weight correction-rule that considers the whole neural structure. This rule was calculated by minimizing a cost function which represents a measure of the overall classification value and allows for the use of a probabilistic interpretation of the neural structure. It means that a father node can update its weight connections on the basis of the classification performed by its children by providing a better cost value; this improvement often corresponds to a reduction in the tree dimension. A supervised algorithm called *Trio Learning,* which trains binary NTs by taking care of sets of nodes instead of single ones, has been proposed by D'Alche'-Buc *et al.* [19]. This learning algorithm trains a node by taking into account the classification performed by its two children nodes. The authors demonstrate that this learning scheme leads to a significant reduction in the tree complexity and increases the generalization power. The GNT model extends the trio learning algorithm to n levels. In addition, it adds some important and innovative characteristics: 1) the GNT model can work with multiclass problems; 2) the GNT learning rule performs an overall optimization of the network that produces smaller trees (which may pose a better generalization capability); 3) the application of a normalization procedure to the activation values of each node of the GNT allows a probabilistic interpretation to be associated to the classification path, which provides better classification performances with respect to classical NT models; and 4) the classification phase can exploit the probabilistic interpretation, and provide, as boundary regions, hypersurfaces that fit the pattern distributions better then hyperplanes.

This paper is organized as follows: The standard NTs architecture and the related learning algorithm are briefly outlined in Section II. In Section III, the GNT model is presented. Specifically, the weight correction-rule and the stability criteria are described. Experimental results on both artificial geometric pattern distribution (e.g., double spiral, etc.) and real data are presented in Section IV. Some rate-classification comparisons between GNT and classic neural models are also given.

II. NT Model

NTs are NNs with a tree topology in which each node is a simple perceptron [5]–[9], [14]. They adopt a supervised training algorithm where two functional phases can be distinguished: the training phase and the classification phase.

A. Training Phase

The learning algorithm calculates the tree structure as well as the connection's weight for each node. For an n-class problem, n neurons are used in each node. A binary vector is used as local encoding scheme, where the class i is labeled with 1 in the ith bit and all the other bits are 0. Learning is executed by minimizing a cost function such as the mean square error (SME) or other error functions [4]–[6]; for instance, Sankar and Mammone [6] adopted a $\mathcal{L}1$ norm, where they demonstrate it permits the reduction in the number of outliers. At the end of the learning phase, each node is responsible for dividing the feature space into n convex regions.

The NTs learning algorithm may be summarized as follows.

1) The patterns of the training set (TS) are presented to a node (at the beginning they are presented to the root which is unique) that is trained until a stop condition is verified. The training consists in dividing the training set (problem space) into more subsets (subregions) [Fig. 1(a)].
2) For those subsets (regions) that are homogeneous (they contain all patterns of the same class), the process is completed. Each of these subsets (regions) is labeled with the class of the patterns that are contained in it and the related node is called a *leaf node* [Fig. 1(b)].
3) If one of the obtained subsets (regions) is not homogeneous, a new node is added to the NT in order to learn the patterns contained in such a subset (region). Go to Step 1 [Fig. 1(b)].
4) Learning stops when all subsets (regions) become homogeneous [Fig. 1(c)].

B. Classification Phase

For the classification task, unknown patterns are presented to the root node. The class is obtained by moving down the tree. Starting from the root, the activation value of the current node provides the next node to be considered until reaching a leaf node that assigns the class of the input pattern. Each node applies the "winner-takes-all" rule so that

$$\boldsymbol{x} \in \text{ class } i \Longleftrightarrow f(\boldsymbol{w}_j \cdot \boldsymbol{x}) \leq f(\boldsymbol{w}_i \cdot \boldsymbol{x}) \qquad \forall\, j \neq i$$

where f is the activation function, $\boldsymbol{w}_i$ is the vector of weights of the connections from inputs to the ith output, and $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ is the input vector. This rule determines the classes associated with the leaf nodes as well as the paths on the internal nodes. Fig. 2 shows an example of the classification of a three-class problem in a four-dimensional feature space.

III. GNT Model

As in the classical NT model [5], the GNTs structure consists of a set of units organized into a tree. Each node of the GNT is composed of two parts: the single-layer neural network, and the normalizer, which has to make an output sum equal to one (Fig. 3). Let $\boldsymbol{o} = (o1, o2, \ldots, o_n)$ be the output vector. The normalized vector is defined as $\boldsymbol{nr} = (nr_1, nr_2, \ldots, nr_n)$ where

$$nr_i = \frac{o_i}{\sum\limits_{l=1}^{n} o_l}. \tag{1}$$

The normalized vector allows a probabilistic interpretation to be given of the classification made by the GNT on the input pattern $\boldsymbol{x}$. For example, let us suppose we classify a pattern $\boldsymbol{x}$, as being a binary tree composed of a father and two children nodes, where $nr_1 = 0.2$ and $nr_2 = 0.8$ on the root node; $nr_1 = nr_2 = 0.5$ on the left child, and $nr_1 = 0.3$ and $nr_2 = 0.7$ on the right child. Let C_2 be the correct class of $\boldsymbol{x}$, then $Pr(C_2|\boldsymbol{x}) =$

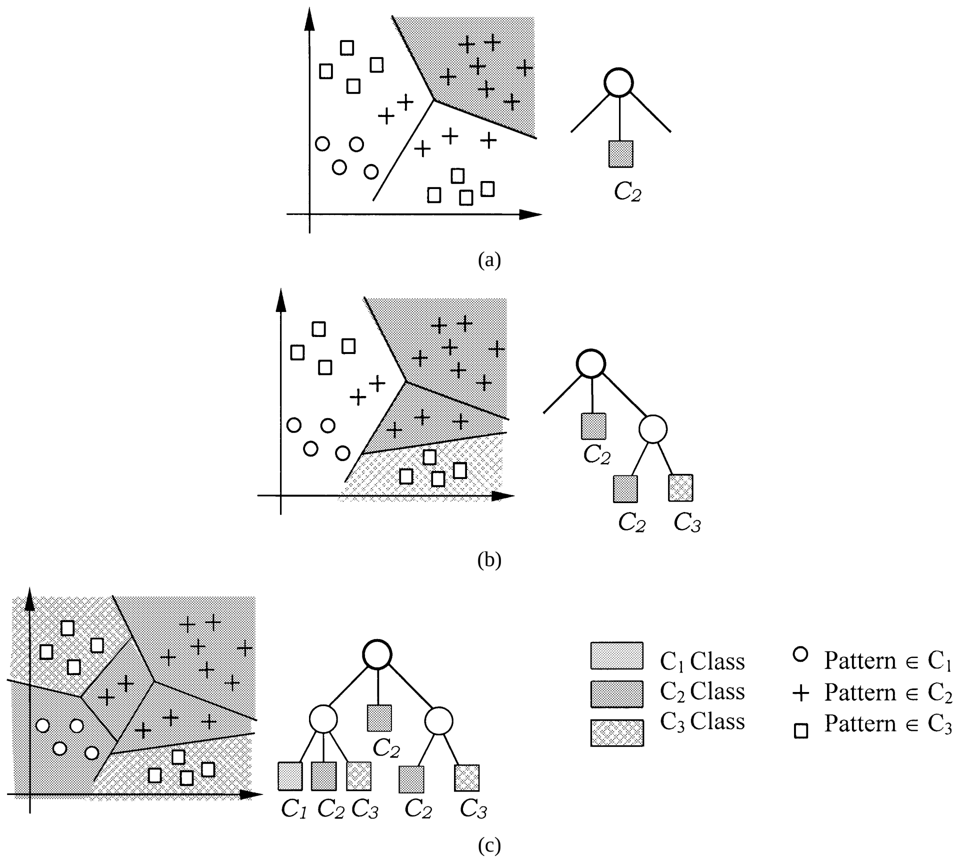


Fig. 1. Geometric interpretation of the NT learning algorithm. (a) Partition of the feature space due to the root node. (b) Partition of the feature space due to internal nodes. (c) Final tree configuration.

$(0.2 \cdot 0.5 + 0.8 \cdot 0.7) = 0.66$ is the *a posteriori* probability needed to obtain the class C_2 given the pattern $\boldsymbol{x}$. Obviously, $Pr(C_1|\boldsymbol{x}) = 0.2 \cdot 0.5 + 0.8 \cdot 0.3 = 1 - Pr(C_2|\boldsymbol{x}) = 0.34$. The *a posteriori* probability $Pr(C_2|\boldsymbol{x})$ could be considered as a kind of goodness measurement of the classification performed by the tree. In general, we can define this measurement for a generic subtree rooted on node l

$$PP_l(C_x|\boldsymbol{x}) = \sum_{f \in F_l} P_{f(x)} \tag{2}$$

where F_l is the set of leaves which corresponds to the correct class of $\boldsymbol{x}$ (i.e., C_x) and $P_{f(\boldsymbol{x})}$ is the probability associated with the path from node l to the generic leaf f belonging to F_l. Therefore, we can write $Pr(C_x|\boldsymbol{x}) = PP_{\mathrm{root}}(C_x|\boldsymbol{x})$.

A. Weight Updating Rule

In NTs, learning is local in the sense that each node is trained only with patterns which stay in its convex region. At the end of the learning (when all patterns are correctly classified by the node or when a condition of minimal error is reached) the node weights are definitively fixed. In contrast, when the GNTs learning rule is applied to a generic internal node, the entire training set is considered. In fact, as a consequence of the updating rule which will be presented further, a generic internal node is affected by all patterns. As a result, a father node can update its weight connections on the basis of the classification performed by its children and, consequently, to provide a better probabilistic classification. Experimental tests (see Section IV) have demonstrated that this improvement generally corresponds to a reduction in tree dimension.

As with MLPs, the weight-updating rule is obtained by minimizing an error function through the gradient descent method. On feedforward networks, the error function is usually represented by the square of the difference between the obtained and the desired outputs. In the GNT learning, the same approach is followed, and the following error function E has been selected:

$$E(\boldsymbol{x}) = \frac{1}{2} \cdot [PP_{\mathrm{root}}(C_x|\boldsymbol{x}) - 1]^2 \tag{3}$$

where $PP_{\mathrm{root}}(C_x|\boldsymbol{x})$ represents the output obtained when the pattern $\boldsymbol{x}$ is presented to the network, whereas the desired output is set to one (the maximum of the probability).

Let us consider the generic node l placed on a generic position in the tree. Let $l_1, l_2, \ldots, l_n$ be its children nodes. Let $PP_i = PP_{l_i}(C_x|\boldsymbol{x})$ be the probabilities related to the subtrees rooted on $l_i (i = 1, \ldots, n)$. If $Path(l)$ is the product of the probability values of all branches from the root to the node l, $Path(l) = \prod_{s=1}^{l} p_s$ and $o_i = (1/1 + e^{-\mathrm{net}_i})$ is the sigmoidal activation function where $\mathrm{net}_i = \sum_{j=1}^{m} w_{ij} \cdot x_j$, then, the updating weight rule can be obtained as follows:

$$\frac{\partial E(\boldsymbol{x})}{\partial w_{ij}} = [PP_{\mathrm{root}}(C_x|\boldsymbol{x}) - 1] \cdot \sum_{r=1}^{n} \left[\mathrm{Path}(l) \cdot PP_r \cdot \frac{\partial nr_r}{\partial o_i} \cdot o_i \cdot (1 - o_i) \cdot x_j \right] \tag{4}$$

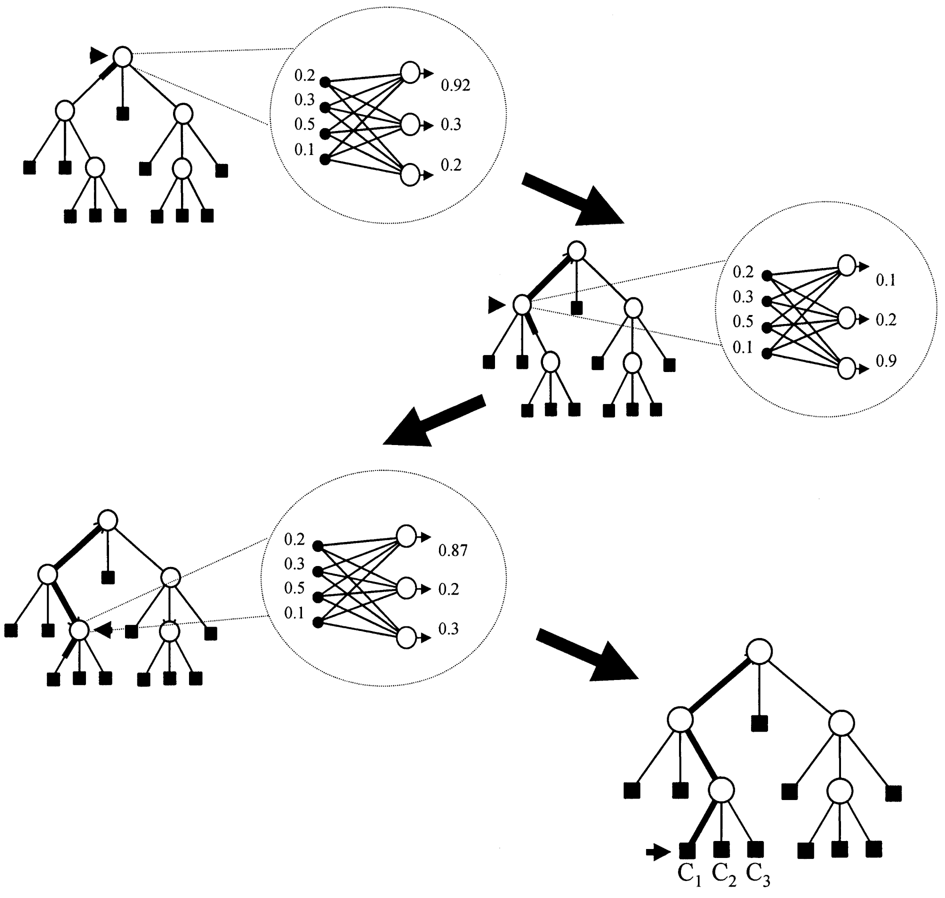


Fig. 2. Classification example of a three-class problem. The tree classifies the input pattern by following the path from the root to a leaf node according to the winner-takes-all rule.

where

$$\frac{\partial nr_r}{\partial o_i} = \frac{\sum_{q\neq i}^{n} o_q}{\left(\sum_{q=1}^{n} o_q\right)^2}, \qquad \text{when } r = i$$

$$\frac{\partial nr_r}{\partial o_i} = -\frac{o_i}{\left(\sum_{q=1}^{n} o_q\right)^2}, \qquad \text{when } r \neq i.$$

The proof for (4) is provided in Appendix A. Leaf nodes are updated with classical methods applied to single-layer networks (e.g., minimizing the LMS error by a gradient descend). This implies that unlike internal nodes, leaf nodes consider only the patterns of their convex region. In order to update the weights of the node l, the GNT algorithm must know the probability values of its children, $PP_i (i = 1, \dots, n)$. In particular, to update the weights of an internal node l, the following information is needed (4): 1) the *a posteriori* probability (obtained by the classification) of pattern $\boldsymbol{x}$ presented to the tree; 2) the value of Path(l); and 3) the PP_r values of the children nodes r of l. The GNT training algorithm proceeds through a recursive procedure that implements, as in MLPs, a feedforward phase and a backpropagation phase. The first allows the calculation of the classification values on the leaves, while the second calculates the PP_r values needed for the updating of the nodes.

In the following paragraphs, the learning algorithm and the stability criterion will be described.

B. Learning Algorithm

The learning algorithm consists of training, level by level, the tree structure beginning with the leaves and proceeding toward the root. For example, if we consider working with a tree that has a depth of k, first all of the leaves (at depth k) have to be trained, and when the PrM reaches a stability value, one epoch of training is executed at level $(k-1)$. At this point, a convex region corresponding to a leaf node could be modified and some patterns could be moved from this region to another, or vice versa: if this happens, then it is necessary to restart the training from level k. When the stability is once again achieved, another learning epoch will be executed at level $(k-1)$, and so on. When the same stability condition is reached at level $(k-1)$, level $(k-2)$ is considered. When this happens at the root, we may say that the current tree structure is not able to learn the whole TS and a new level of nodes is added to the tree.

This learning technique has been applied due to the fact that after observing several experimental tests, we noted that updating the whole structure (all layers at once) causes instability in the learning process and results in unsatisfactory improvements of classification. The learning rule of a generic node l uses the classification results of the l children nodes, therefore it is not worth updating the weights of l if its children have just

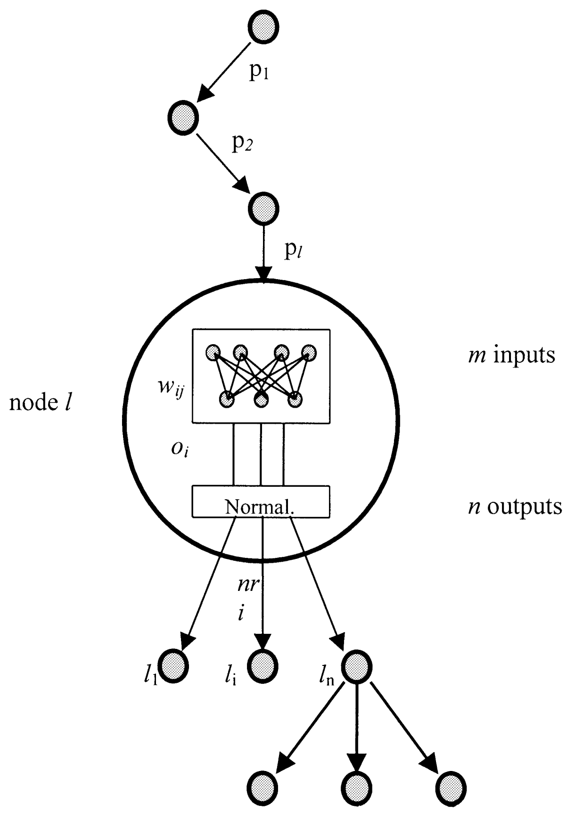


Fig. 3. General scheme of a generic l node and its position into the GNT.

started their learning. The classification boundaries must proceed gradually toward a better position; only when a subtree reaches a stable condition (with a level by level updating), we can try to improve the structure by working on the parent node. In other words, there is not much sense in updating a node if its children have not obtained a "good classification" condition.

C. Stability Criteria

As with in NTs, GNTs also require some stop-training criteria. Two stability criteria have been used. The first consists in verifying the correct classification of the entire training set, which is tested at each training epoch; essentially, it corresponds to the stop learning criteria. The second criteria establishes how to modify the tree structure and consists in testing the mean-classification probability PrM

$$PrM = \frac{\sum_{x \in TS} PP_{\text{root}}(C_x|\boldsymbol{x})}{N} \tag{5}$$

where N is the cardinality of the training set.

The second stability condition is satisfied when PrM does not increase up to *toler* (a real value ranging between zero and one) after *wait* (an integer value) consecutive training epochs. When this occurs on the root node, we say that the current tree structure is not able to learn the whole TS, thus a new level of nodes (which will be the new leaves) is added to the tree. Of course, such an operation allows the attainment of smaller convex subregions that better fit the TS distribution. The selection of the *wait* and *toler* parameters is very important for speeding up the learning process; in fact, the choice of tight values (*toler* near to zero and high values of *wait*) could require long training times. For example, if the training set has got a particular distribution which is impossible to learn with a k-level tree (it needs at least $k+1$ levels), it will be too expensive to employ such values. *Wait* and *toler* parameters determine the stability condition, thus they are closely correlated with the "backtracking frequency": tight values imply rare "backtracking" steps toward the upper nodes and, hence, a longer search time of the hyperplane positions in the feature space. On the contrary, the choice of wide values (*toler* near to one and low values of *wait*) produces a fast training, but also a lesser optimization, and in general, a structure having more nodes. We can regulate these parameters depending on time availability, from the quickest training, which corresponds to the NTs performance, to the longest, which corresponds to a good optimization.

D. Classification Algorithm

The GNTs classification algorithm is similar to that used by NTs: the class of the unknown pattern is obtained by going up the tree, starting from the root and reaching a leaf node according to a "winner-takes-all" rule. By using this approach, the classes are associated to convex regions which have boundaries represented by parts of hyperplanes (see Section IV, paragraph A describing experiments on synthetic TS). Exploiting the probabilistic interpretation, a GNT can classify a pattern also by its *a posteriori* probability $PP_{\text{root}}(C_x|\boldsymbol{x})$, where C_x in this case is the class which obtains the highest value. This simple strategy produces boundaries that in general can be hyper surfaces. The best results on real experiments have been obtained with this technique.

IV. Experimental Results

In this section, some experimental results on both synthetic and real data are presented. Classification comparisons among the GNT, classical NTs [6] and MLPs are also proposed.

A. Synthetic Training Set

Tests on artificial pattern distributions provide an easier way to compare performance classifications over a two-dimensional (2-D) feature space; the behavior of different learning algorithms may be observed graphically.

The first test is characterized by a training set of 64 patterns distributed on a 2-D feature space, as in Fig. 4(a). Fig. 4(b) shows the results of the NT learning algorithm visualized through the classification of the square area which contains the TS (the NT structure is composed of 20 nodes). The same TS is applied to learn the GNT: it produces the classification shown in Fig. 4(c).

It may be observed that: 1) the decision regions obtained by the GNT show a better representation of the geometric distribution of the training set and 2) the complexity of the tree structure is reduced, i.e., only four nodes are needed versus the 20 nodes required by the NT. Finally, the same TS is used to train an MLP; the classification results are shown in Fig. 4(d). The structure of the MLP is composed of five input nodes, a hidden layer, and two output nodes. The number of neurons

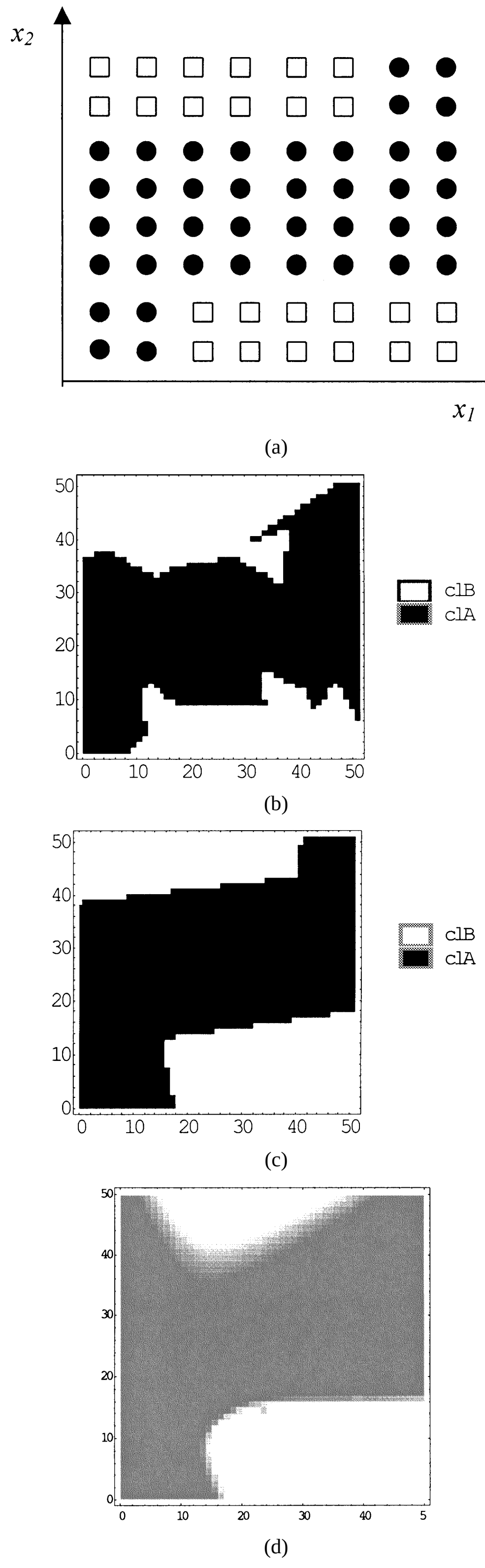


Fig. 4. (a) Set of 64 patterns distributed on a 2-D feature space. (b) Classification obtained with an NT (depth = 6) composed of 20 nodes (three are split nodes). (c) Classification obtained by a GNT (depth = 3) composed of six nodes. (d) Classification obtained by an MLP, composed of two input nodes, two output nodes, and four neurons in the hidden layer.

in the hidden layer was obtained by trials: seven neurons have achieved the best result.

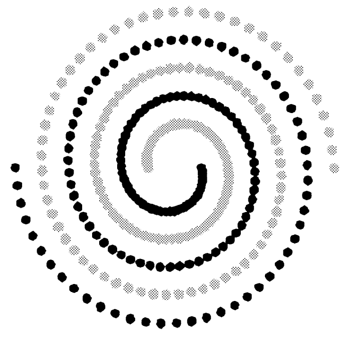
B. Problem of the Double Spiral

In the next experiment, the double spiral problem is taken into consideration (Fig. 5). This problem is deemed a difficult task because of the complexity of learning the boundary regions of the TS. The selected TS consists of two spirals (i.e., a two class problem); each one composed of 150 patterns [Fig. 5(a)]. Fig. 5(b) represents the classification obtained by the classical NT. The tree generated has 102 nodes distributed on nine levels. Many irregularities in the shape of the classified regions can be observed. In Fig. 5(c), the GNT classification is shown. The tree structure is composed of 48 nodes distributed on eight levels. This result was obtained after a sequence of a few trials, taking into account a tradeoff between training times and performances. It is important to note that the training of the MLP was very time-consuming and unsuccessful: after about 20 attempts at training, beginning with the simplest network and growing it, a satisfactory classification was not achieved.

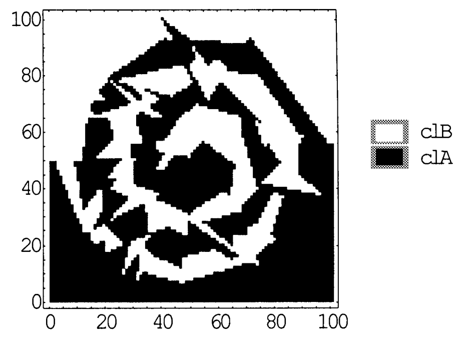
C. Real Training Set—Phoneme Database

A real database (the Phoneme database) containing several thousand multidimensional real patterns has been considered. This database was developed in the European ROARS (RObust Analytical speech Recognition System) ESPRIT project [20]. The aim of this project was the development of a real-time analytical system for French speech recognition that was able to distinguish between nasal and oral vowels. The TS is composed of 5404 patterns distributed over two classes: 3818 of class 0 type and 1586 of class 1 type. A pattern is described as having five features, i.e., a five-dimensional space. For the evaluation of the classification performance of the Phoneme database, the methodology introduced in the ELENA project [21] has been used; indeed, we exploited ELENAs results for a more exhaustive comparison between various classifiers. These are the k-nearest neighbor (k-NN) classifier [22], selected for its powerful probability density estimation properties, the Gaussian quadratic classifier (GQC) [22], the most classical statistical parametric simple classification method, the learning vector quantizer (LVQ) [22], a powerful nonlinear iterative learning algorithm proposed by Kohonen, the reduced Coulomb energy (RCE) algorithm [22], an incremental region of influence algorithm, the inertia rated vector quantizer (IRVQ) and the piecewise linear separation (PLS) classifiers [21], developed within the framework of the Elena project. The test used for comparison is the holdout set, averaged over five different partitions of the original database in two independent learning sets and test sets, each containing half of the total amount of the available patterns. The averaged mean error computed on the confusion matrixes is calculated based on its 95% confidence interval. Results are shown in Fig. 6. The GNT classifier obtains a percentage of 15.19, $\pm 1.35\%$ for an error rate.

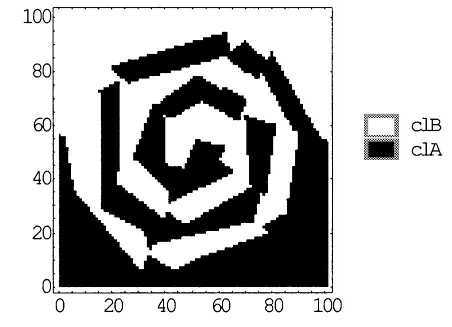
Since the Phoneme database has a high degree of overlapping (in fact the error calculated with k-NN is quite high), the learning process requires a high computational time. In order to reduce such time, a forward pruning technique has been employed. It simply consists of saving, in a "pocket," the best structure that provides the higher value of G parameter. G combines the PrM information with the number Nc of patterns correctly



(a)



(b)



(c)

Fig. 5. (a) Double spiral training set. (b) Classification obtained with an NT (depth $= 9$) with 102 nodes (two split nodes). (c) Classification obtained by a GNT (depth $= 9$) with 48 nodes.

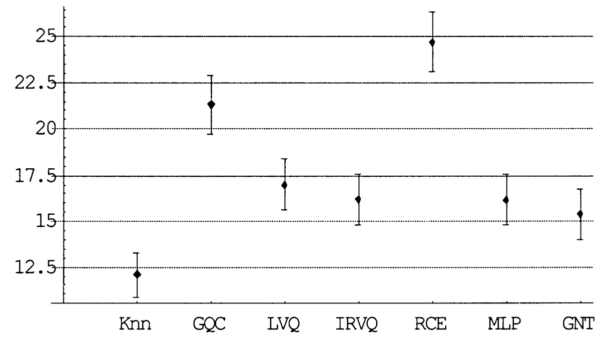


Fig. 6. Averaged error rate with a 95% confidence interval obtained on the Phoneme TS by the following classifiers: k-NN, GQC, LVQ, IRVQ, RCE, MLP, and GNT. The k-NN obtains the best performance, but its classification process requires a very long time. The GNT model obtains the second best performance, but with real-time classification.

classified: $G = \alpha(Nc/N) + (1-\alpha)PrM$; where α has been fixed at 0.5 in all tests. Let us assume that during the learning of the tree, composed of K layers, the best G was $MaxG$. If, by the end of the learning of the $(K+1)$ layer-tree, G has never been greater than $MaxG$, then the learning process stops and the best structure (stored in the pocket) is considered. This is a quite simple heuristic method and not an optimal pruning technique. However, it is easy to implement, it allows a reduction in the learning time, and it needs no further parameter introduction. Training takes, on average, about 1 h 50 min utilizing a Pentium III 400-MHz processor.

V. Conclusion

Like other constructive algorithms, a GNT grows the tree structure in accordance with the classification problem presented (i.e., TS). Growing the smallest NT that correctly classifies the training data set is an NP-hard problem; such problems are usually confronted with heuristics methods that do not guarantee the optimal solution but provide a good solution within an acceptable time. The GNT allows the updating of the weight connections of all the nodes while taking care of the classification results of the lower nodes. It has demonstrated to be very effective in learning training sets with complex decision boundaries, as we saw with the double spiral problem and, unlike SAINT and SPA algorithms, it also shows good classification performance with respect to the MLP.

Although the GNT works by maximizing the mean probability of classification, it is interesting to observe that this operation corresponds to the growing of small trees having good generalization properties. The parameter regulation is a straightforward way to control the tree optimization degree versus the learning time. Learning processes requiring a very long time generally allow us to obtain smaller trees with a high generalization power, whereas a shorter training time produces bigger structures. The probabilistic interpretation that allows the classification of patterns by the *a posteriori* probability provides very good results in the real Phoneme database.

Appendix

In this example, the node label l is omitted because we always refer to node l.

Proof

Let $E(\boldsymbol{x}) = 1/2 \cdot [PP_{\text{root}}(C_x|\boldsymbol{x}) - 1]^2$ be the error function associated to the pattern $\boldsymbol{x}$. Partial derivatives with respect to weights w_{ij} are

$$\frac{\partial E(\boldsymbol{x})}{\partial w_{ij}} = \frac{\partial E(\boldsymbol{x})}{\partial PP_{\text{root}}(C_x|\boldsymbol{x})} \cdot \frac{\partial PP_{\text{root}}(C_x|\boldsymbol{x})}{\partial w_{ij}} \tag{a1}$$

where

$$\frac{\partial E(\boldsymbol{x})}{\partial PP_{\text{root}}(C_x|\boldsymbol{x})} = PP_{\text{root}}(C_x|\boldsymbol{x}) - 1 \tag{a2}$$

$$\frac{\partial PP_{\text{root}}(C_x|\boldsymbol{x})}{\partial w_{ij}} = \sum_{r=1}^{n} \frac{\partial PP_{\text{root}}(C_x|\boldsymbol{x})}{\partial nr_r} \cdot \frac{\partial nr_r}{\partial w_{ij}}. \tag{a3}$$

Regarding Fig. 3, we can observe that by changing one weight, w_{ij} produces the variation of the output o_i and hence, the variation of all nr_i outputs due to the normalizer. Let us calculate the variation of the probability $PP_{\rm root}(C_x|\boldsymbol{x})$ with respect to the variation of the normalized output nr_i, i.e., $(PP_{\rm root}(C_x|\boldsymbol{x})/\partial nr_r)$. The value of $PP_{\rm root}(C_x|\boldsymbol{x})$ can be calculated as follows:

$$PP_{\rm root}(C_x|\boldsymbol{x}) = \sum_{f\in F} Pf(\boldsymbol{x}) \tag{a4}$$

where F is the set of leaves, which corresponds to the correct class C_x, and $P_{f(x)}$ is the probability associated to the path from the root to the generic leaf node f , belonging to F. Only the paths that pass through nr_i will be considered in the nr_i variation. They have the first part of the path in common $Path(l)$, while the second part (that is, under node l) belongs to the subtree having l_i as root. Having previously defined PP_i as the probability associated to the ith child of l, we can write

$$\frac{\partial PP_{\rm root}(C_x|\boldsymbol{x})}{\partial nr_i} = \frac{\sum\limits_{f\in F} P_{f(x)}}{\partial nr_i} = Path(l)\cdot PP_i. \tag{a5}$$

Let us consider the variation of the rth normalized output with respect to the variation in the weight w_{ij}, $(\partial nr_r/\partial w_{ij}) = (\partial nr_r/\partial o_i)\cdot(\partial o_i/\partial w_{ij})$. Referring to $nr_r = o_r/(\sum_{q=1}^{n} o_q)$, we can easily obtain the nr derivatives with respect to the ith output:

$$\frac{\partial nr_r}{\partial o_i} = \frac{\sum\limits_{q\neq i}^{n} o_q}{\left(\sum\limits_{q=1}^{n} o_q\right)^2}, \qquad \text{when } r = i \text{ and}$$

$$\frac{\partial nr_r}{\partial o_i} = -\frac{o_i}{\left(\sum\limits_{q=1}^{n} o_q\right)^2}, \qquad \text{when } r \neq i. \tag{a6}$$

It is important to observe that a positive variation of the o_i output produces a positive variation on nr_i and a negative variation on nr_r when $r \neq i$ (vice versa for negative variations). This aspect has a great importance on learning as it maintains the correct probabilistic structure.

The variation of the ith output with respect to w_{ij} is $(\partial o_i/\partial w_{ij}) = o_i\cdot(1-o_i)\cdot x_j$, so that on the whole structure we obtain

$$\frac{\partial E(\boldsymbol{x})}{\partial w_{ij}} = [PP_{\rm root}(C_x|\boldsymbol{x}) - 1] \cdot\sum_{r=1}^{n}\left[Path(l)\cdot PP_r\cdot\frac{\partial nr_r}{\partial o_i}\cdot o_i\cdot(1-o_i)\cdot x_j\right]. \tag{a7}$$

It can be written as follows by keeping the terms that are independent from r out of the sum:

$$\frac{\partial E(\boldsymbol{x})}{\partial w_{ij}} = [PP_{\rm root}(C_x|\boldsymbol{x}) - 1]\cdot Path(l)\cdot o_i\cdot(1-o_i)\cdot x_j \cdot\sum_{r=1}^{n}\left[PP_r\cdot\frac{\partial nr_r}{\partial o_i}\right]. \tag{a8}$$

Finally, the term PP_l can be expressed recursively from its children values $nr_1,\ldots,nr_n$

$$PP_l = \begin{cases} \sum\limits_{r=1}^{n} nr_r\cdot PP_r, & \text{when } l \text{ is an internal node} \\ nr_{C_x}, & \text{when } l \text{ is a leaf node.} \end{cases}$$

References

[1] S. Rasoul and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, June 1991.

[2] C. Lau and B. Widrow, "Special issue on neural networks," *Proc. IEEE*, vol. 78, Sept. 1990.

[3] P. E. Utgoff, "Perceptron tree: A case study in hybrid concept representation," in *Proc. 7th Nat. Conf. Artificial Intell.*, Saint Paul, MN, 1988, pp. 601–606.

[4] J. A. Sirat and J.-P. Nadal, "Neural trees: A new tool for classification," *Network*, vol. 1, pp. 423–448, 1990.

[5] M. Golea and M. Marchand, "A growth algorithm for neural network and decision trees," *Europhys. Lett.*, vol. 12, no. 3, pp. 205–210, 1990.

[6] A. Sankar and R. J. Mammone, *Neural Tree Networks, Neural Network: Theory and Application*, R. Mammonee and Y. Zeevi, Eds. New York: Academic, 1991.

[7] ——, "Optimal pruning of neural tree networks for improved generalization," in *Proc. Int. Joint Conf. Neural Networks*, Seattle, WA, July 8–12, 1991, pp. 809–814.

[8] ——, "Growing and pruning neural tree networks," *IEEE Trans. Comput.*, vol. 42, pp. 291–299, Mar. 1993.

[9] M. G. Rahim, "A neural tree network for phoneme classification with experiments on the TIMIT database," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, San Francisco, CA, Mar. 23–26, 1992, pp. 345–348.

[10] K. R. Farrell, R. J. Mammone, and K. T. Assaleh, "Speaker recognition using neural networks and conventional classifiers," *IEEE Trans. Speech Audio Processing*, vol. 2, pp. 194–205, Jan. 1994.

[11] I. K. Sethi, "Entropy nets: From decision trees to neural networks," *Proc. IEEE*, vol. 78, pp. 1605–1613, Oct. 1990.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth and Brooks.

[13] M. Lehtokangas, "Cascade-correlation learning for classification," *IEEE Trans. Neural Networks*, vol. 11, pp. 795–798, Mar. 2000.

[14] X. O. Tang, "Multiple competitive learning network fusion for object classification," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 532–543, Aug. 1998.

[15] M. Pontil and A. Verri, "Support vector machines for 3D object recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, pp. 637–646, June 1998.

[16] T. Li, Y. Y. Tang, and L. Y. Fang, "A structure parameter adaptive (SPA) neural tree for the recognition of large character set," *Pattern Recognition*, vol. 28, no. 3, pp. 315–329, 1995.

[17] S. Behnke and B. Karayiannis, "Competitive neural trees for pattern classification," *IEEE Trans. Neural Networks*, vol. 9, pp. 1352–1369, Nov. 1998.

[18] H. H. Song and S. W. Lee, "A self-organizing neural tree for large-set classification," *IEEE Trans. Neural Networks*, vol. 9, pp. 369–380, May 1998.

[19] F. D'Alché-Buc, D. Zwierski, and J. P. Nadal, "Trio learning : A new strategy for building hybrid neural trees," *Int. J. Neural Syst.*, vol. 5, no. 4, pp. 259–274, 1994.

[20] P. Alinat, "Thompson Tech. Rep. ASM 93/S/EGS/NC/079 of ROARS ESPRIT II Project 5516," Tech. Rep, 1993.

[21] F. Blayo, Y. Cheneval, A. Guerin-Dugue, R. Chentouf, C. Aviles-Cruz, J. Madrenas, M. Moreno, and J. Voz, "Deliverable r3-b4-p Task b4: Benchmarks, Tech. Rep. ESPRIT Basic Research Project 6891, Enhanced Learning for Evolutive Neural Architecture," Tech. Rep, 1995.

[22] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.