Growing and Pruning Neural Tree Networks

Ananth Sankar, *Member, IEEE*, and Richard J. Mammone, *Senior Member, IEEE*

***Abstract*—A new pattern classification method called Neural Tree Networks (NTN) is presented. The NTN consists of neural networks connected in a tree architecture. The neural networks are used to recursively partition the feature space into subregions. Each terminal subregion is assigned a class label which depends on the training data routed to it by the neural networks. In decision tree algorithms, the feature space is usually partitioned by using hyperplanes that are constrained to be perpendicular to the feature axes. The use of a neural network at each tree node, as in the NTN, results in better classification performance. The NTN is grown by a learning algorithm as opposed to Multi-Layer Perceptrons (MLP) where the architecture must be specified before learning can begin. This is a drawback of the MLP approach since, in many practical problems, numerous trials are needed before an appropriate architecture can be found. The NTN also provides an attractive tradeoff of classification speed versus hardware implementation as compared to MLP's.**

Growing the smallest NTN that will correctly classify the training data is an NP-complete problem as is the problem of training an MLP. In this paper, a heuristic learning algorithm based on minimizing the L1 norm of the error is used to grow the NTN. We show that this method has better performance in terms of minimizing the number of classification errors than the squared error minimization method used in backpropagation. An optimal pruning algorithm is given to enhance the generalization of the NTN. Simulation results are presented on Boolean function learning tasks and a speaker independent vowel recognition task. It is shown that the NTN compares favorably to both neural networks and decision trees.

***Index Terms*—Decision tree, minimum error classification, neural networks, neural tree networks, pruning.**

I. Introduction

The problem of pattern classification can be stated as follows: Given a set of training feature vectors $\boldsymbol{x}_i$, each with an associated class label $\boldsymbol{y}_i$, find a classification system that will produce the correct label $\boldsymbol{y}_i$ for any feature vector $\boldsymbol{x}_i$ drawn from the same source as the training data. The classification system is usually represented by parameters which are determined by a training algorithm that uses specific training data. These algorithms are broadly classified as supervised and unsupervised algorithms. In supervised training algorithms, the class label is provided with each training vector $\boldsymbol{x}_i$. Examples of such algorithms include the backpropagation algorithm [1]–[3] to train MLP's and decision tree algorithms such as CART [4]. In unsupervised training, the class label is not provided with every training vector. Such algorithms learn the distribution of the source using clustering techniques [5], [6].

Neural Networks, particularly Multi-Layer Perceptrons (MLP), have recently been found to be successful for various classification tasks [1], [7]–[9]. The basic building block for MLP's is a neuron which calculates a nonlinear function of the weighted sum of its inputs. Thus the neuron can be specified by its weights and the nonlinear activation function. In an MLP these neurons are arranged in layers such that the outputs of one layer are connected to the inputs of the next layer. The input feature vector is fed to the first layer of neurons whose outputs become the inputs for the second layer of neurons and so on. Finally, the output of the last layer is the output of the MLP. The last layer is called the output layer and all other layers are called hidden layers. An n-layer network is one that has $n-1$ hidden layers and one output layer. The weights of an MLP are typically found by supervised training algorithms. The problem of training an MLP is NP-complete [10] and therefore all existing algorithms are heuristics. Currently the most popular algorithm is backpropagation [1]–[3], which is essentially a gradient descent procedure on an error surface specified in the weight space. For networks with hidden layers, this error surface is nonconvex, and results in local minima. In addition, the exact number of hidden neurons and the connectivity between layers must be specified before learning can begin. In practice, however, one cannot guarantee that backpropagation will find the correct weights for a given number of neurons and a particular training set. Thus the network architecture must be determined by trial and error.

Decision trees use a sequential decision making strategy to classify a feature vector. At each internal node of a decision tree a test is evaluated to decide which child node the feature vector will be sent to. A commonly used test is the hyperplane test [4]. In this method, the feature vector $\boldsymbol{x}$ is tested with respect to a hyperplane. Depending on which side of the hyperplane it is on, the feature vector is sent to one or the other child node. This results in a binary tree structure. The leaves of the decision tree are used to classify the feature vector. Training a decision tree consists of finding the hyperplanes at the tree nodes. The traditional approach has been to first generate a set of possible hyperplanes and then exhaustively search this set to find the best hyperplane with respect to some distortion metric [4]. In most decision tree algorithms, the hyperplanes are constrained to be perpendicular to the feature space axes. This is very restrictive and can result in a large number of hyperplane tests, even in a linearly separable problem, if the separating hyperplane is not perpendicular to the feature space axes. The CART decision tree algorithm

Manuscript received February 6, 1991; revised August 26, 1991 and March 26, 1992. This work was supported by the New Jersey Commission on Science and Technology and the Center for Computer Aids for Industrial Productivity at Rutgers University.

A. Sankar is with AT&T Bell Laboratories, Murray Hill, NJ 07974. This work was done while he was at the CAIP Center, Rutgers University, Piscataway, NJ.

R. J. Mammone is with CAIP Center and Department of Electrical Engineering, Rutgers University, Piscataway, NJ 08855-1390.

IEEE Log Number 9207181.

[4] does allow for nonperpendicular hyperplanes, but the algorithm used is computationally expensive due to the large search space of hyperplanes that need to be evaluated.

In this paper we introduce a new approach to pattern classification called Neural Tree Networks (NTN). The NTN is a tree with a simple neural network at each of its nodes. The neural network provides a natural tool for implementing hyperplane tests that are not restricted to be perpendicular to the feature space axes as in decision trees. The weights of the neural networks are updated using a supervised learning algorithm which is more efficient than the exhaustive search techniques of decision trees. The NTN is not necessarily a binary tree as is the case in many traditional decision trees. We show that the NTN provides an attractive trade off of classification speed against hardware implementation as compared to MLP's. The NTN is grown during the learning process rather than specified *a priori* as in MLP's. As opposed to backpropagation, the NTN learning algorithm always converges to a solution on the training set. The NTN grown by the learning algorithm works very well for the training set. However, in order to enhance the generalization of the NTN, it is desirable to prune the NTN. An optimal pruning algorithm is presented for the NTN. This algorithm is an extension of the pruning algorithm given in [4] for binary decision trees. Simulation results show that the classification performance of the NTN is superior to decision trees and comparable to the best performance obtained by MLP's. However, the NTN is grown by the algorithm, thus offering an advantage over MLP's, which are difficult to train for many practical problems.

The remainder of this paper is organized as follows. The NTN architecture and a new learning algorithm are discussed in Section II. An optimal pruning algorithm for the NTN is given in Section III. In Section IV we present simulation results on boolean function learning tasks and a speaker independent vowel recognition task. In addition, various comparisons of the new algorithm to backpropagation and decision trees is given. Section V gives the summary and conclusion of the paper.

II. Neural Tree Network: Architecture and Learning Algorithm

In this section the Neural Tree Network (NTN) architecture and learning algorithm are given. For the sake of simplicity, the architecture and learning algorithm are first discussed for a two-class problem. The method is then generalized to multiclass problems.

A. The Architecture

The new architecture is described first for the two-class case. This leads to a binary tree structure. The more general multiclass case is discussed in Section II-C. In the multiclass case, the NTN is not a binary tree. The NTN uses a decision tree structure with a neuron at each internal node as shown in Fig. 1. The neuron provides an elegant way to implement hyperplanes that are not constrained to be perpendicular to the feature space axes as they are in the case of decision trees. Consider the feature vector $\boldsymbol{x}$ shown in Fig. 1. The feature vector enters the NTN at the root node. If the output of the neuron at the root node is less than 0.5, then the left branch is taken and if the output is greater than 0.5, the right branch is taken. In the case shown in the figure, the left branch is taken and the neuron at the left child node is used to route the feature vector further down the NTN. Fig. 1 shows the path taken by the feature vector $\boldsymbol{x}$ from the root to a leaf of the NTN. Each leaf node corresponds to a class and the feature vector is classified appropriately. During the training phase, the neuron in the root node operates on the entire training set $\mathcal{X}$. This neuron splits the training data into two subsets according to the following rule:

$$\begin{array}{ll} \boldsymbol{x} \in \mathcal{X}_0 & \text{if } f(\boldsymbol{w}_1 \cdot \boldsymbol{x}) > 0.5 \\ \boldsymbol{x} \in \mathcal{X}_1 & \text{if } f(\boldsymbol{w}_1 \cdot \boldsymbol{x}) \leq 0.5 \end{array} \tag{1}$$

where $\boldsymbol{w}_1$ is the weight vector of the neuron at the root, $\boldsymbol{x}$ is the input vector, and $f(.)$ is the sigmoid activation function of the neuron. The subsets, $\mathcal{X}_0$ and $\mathcal{X}_1$, are further divided by the child nodes. The subsets of training data corresponding to the leaf nodes partition the training set $\mathcal{X}$.

The number of neurons in an NTN required for an arbitrary two-class problem can be computed as follows. From Cover's result [11], we know that any arbitrary dichotomy of $d+1$ or less vectors in d-dimensional space can be realized using a neuron that implements a $d-1$-dimensional hyperplane. If we have N training vectors, then these can be partitioned into $\lceil N/(d+1)\rceil$ sets, each containing $d+1$ or less vectors, using an NTN with $\log_2\lceil N/(d+1)\rceil$ levels. One extra level is required to classify the patterns in the $\lceil N/(d+1)\rceil$ sets (this can always be done from Cover's result [11]), thus resulting in an NTN with $1+\log_2\lceil N/(d+1)\rceil$ levels. Hence, the maximum path length from the root to a leaf node is

$$\text{Path Length } = 1 + \log_2 \lceil \frac{N}{d+1} \rceil$$

which is also the number of time units to classify a feature vector. The above analysis shows the existence of an NTN with a maximum path length as given. It is possible that the number of levels is greater if, for example, each neuron separates only one vector from the rest at each stage. However, in practice we found that the number of levels was quite small. Recently Baum [12] has shown that an MLP with $\lceil N/d\rceil$ hidden neurons can solve any two-class problem. If all the neurons are implemented in parallel, then an MLP will take two time units to classify a feature vector—one time unit to fire the hidden layer neurons and the second to fire the output neuron. On the other hand a sequential implementation would require $\lceil N/d\rceil$ time units to classify a feature vector since all the neurons must be fired in an MLP. The NTN, however, needs only one neuron to be implemented in hardware since, at each stage, only one neuron is fired. A look-up table can be used to store the weights of all the neurons. The output of the neuron along with the current clock state is fed to a logic circuit which controls the address lines to the look-up table to determine which weights are loaded onto it for the next step as shown in Fig. 2. The additional hardware results in a few gate delays for each firing of the neuron, but this time is small compared to the processing time of the neuron

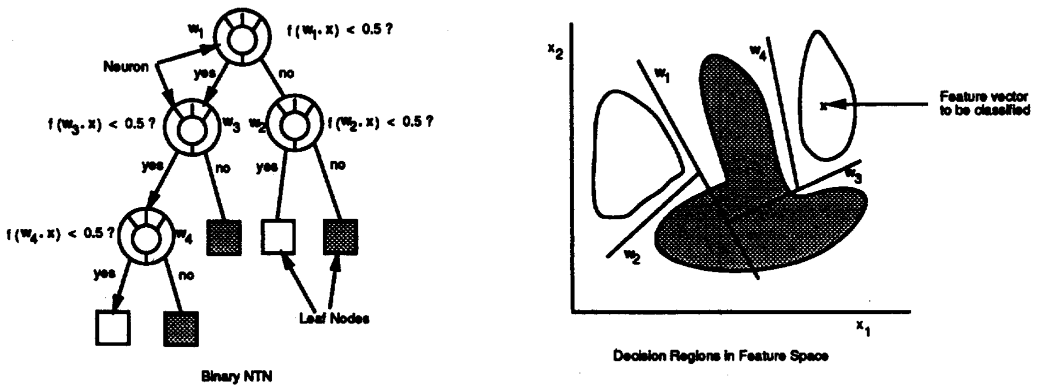


Fig. 1. Binary NTN and corresponding decision regions in feature space.

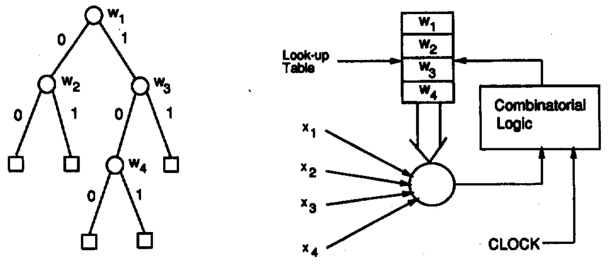


Fig. 2. Implementation of NTN by single processor.

itself. The classification time of the NTN is thus approximately equal to the path length, $1+\log_2\lceil N/(d+1)\rceil$. There is a trade off between the speed of classification and the number of neurons that need to be implemented. If the implementation complexity is defined as the product of the classification time and the number of neurons implemented in hardware, then the complexity of the NTN is

$$C_{NTN} = 1+\log_2\lceil\frac{N}{d+1}\rceil, \tag{2}$$

and the complexity of the MLP is

$$C_{MLP} = 2\lceil\frac{N}{d}\rceil. \tag{3}$$

From these expressions, it is clear that there is a logarithmic relation between C_{NTN} and C_{MLP}, so that

$$C_{NTN} \approx \log_2 C_{MLP}. \tag{4}$$

Clearly the complexity of the NTN is lesser and hence the NTN offers an attractive implementation advantage over the MLP.

B. The Learning Algorithm

As described in the previous section, the NTN successively partitions the training set into subsets, assigning each subset to a different child node. Each division of the training set adds a level to the NTN. It is clearly possible to grow a very large NTN that correctly classifies all the training data. However for good generalization, the number of nodes should be small. Thus it is desired to grow the smallest possible NTN that correctly classifies the training data. This problem has been shown to be NP-complete [13]. Hence we shall solve the problem by using a heuristic approach.

The heuristic used for the NTN will attempt to minimize the number of misclassifications at each internal node of the NTN as opposed to minimizing the squared error of the neuron output. There have been no optimal algorithms for the minimum classification error problem. Some heuristic methods have been suggested for this problem [14]–[17]. The heuristic we developed is called the L1 norm algorithm [15], [18]. In this approach, the neuron is viewed as implementing a sigmoidal surface in feature space. It is required that this surface be at a height of 0 for class 0 patterns and at a height of 1 for class 1 patterns. Thus the error for any pattern $\boldsymbol{x}$ is the difference between the desired height (1 or 0) and the actual height, $y = f(\boldsymbol{w}\cdot\mathrm{x})$, where f is the sigmoid function and $\boldsymbol{w}$ is the weight vector of the neuron. Thus the problem is analogous to one of nonlinear regression where the sigmoid surface is the model. The input patterns that do not fit the model are called outliers. The best fit would be the one with the fewest outliers. Minimizing the absolute errors (L1 norm) rather than the squared errors (L2 norm) results in fewer outliers. Hence, in our algorithm, we minimize the L1 norm of the neuron errors to get a good approximation to the minimum classification error problem.

The L1 error when pattern j is presented to the network is defined by

$$E_j = \mid t_j - y_j \mid \tag{5}$$

where t_j is the required output for the neuron when training pattern j is presented and y_j is the actual output of the neuron. The gradient descent weight update rule for the neurons using the L1 norm error can be shown to be

$$w_k^{n+1} = w_k^n - \eta y_j(1-y_j)\mathrm{sgn}(t_j - y_j)x_k \tag{6}$$

where w_k^n is the weight from the kth input x_k to the neuron at iteration n. The sgn function gives the sign of its argument, and η is a positive step size. The gradient descent algorithm requires that the error function E_j be differentiable at all points. Note that $\mathrm{sgn}(t_j - y_j)$, the derivative of E_j, is not

TABLE I
CLASSIFICATION PERFORMANCE ON NONLINEARLY SEPARABLE BOOLEAN FUNCTIONS

Number of Variables, M	Algorithm	Number correctly classified	Percent correctly classified
10	L1 Norm	592	57.9
	L2 Norm	524	51.2
	Thermal Perceptron	601	58.7
8	L1 Norm	159	62.2
	L2 Norm	134	52.5
	Thermal Perceptron	161	63.1
6	L1 Norm	48	76.2
	L2 Norm	40	62.9
	Thermal Perceptron	49	77.8
4	L1 Norm	12	80.9
	L2 Norm	10	64.9
	Thermal	13	81.3

defined when $t_j = y_j$. However this does not affect the algorithm since no updates are made when the output agrees with the target.

We now compare the performance of the L1 norm algorithm with the standard least squares technique and the thermal perceptron algorithm [16], which has recently been proposed for the minimum classification error problem. The thermal perceptron algorithm is a variation of the perceptron algorithm and uses a temperature controlled step size very much like in simulated annealing [19].

Highly nonlinearly separable M input Boolean problems were generated by randomly assigning an output value of 0 or 1 to each of the 2^M input vectors. The average performance of each of the algorithms over 100 different trials is reported in Table I for different values of M. The table shows that the thermal perceptron algorithm and the L1 norm algorithm have almost the same classification performance. However, the L2 norm algorithm has comparatively poor performance.

The L1 norm algorithm is used to train the neuron in each internal NTN node. The weight updates are stopped when the average error over some time window does not decrease beyond a small threshold. The hyperplane formed by the neuron is then used to split the feature space into two subregions using (1). Each subregion is assigned to a separate child node where the algorithm is then repeated. There are two stopping conditions for the algorithm: 1) The training set in the NTN node contains only one class and so no further splitting is necessary; 2) The neuron is unable to split the training set even though there are two classes. This latter case can occur when there are very few examples from one class surrounded by an extremely large number of examples from the other class resulting in a "donut" shape as shown in Fig. 3. In this case, the minimum misclassification solution would be to classify all patterns into the class with the larger number of training data. We found that this situation very rarely occurs in

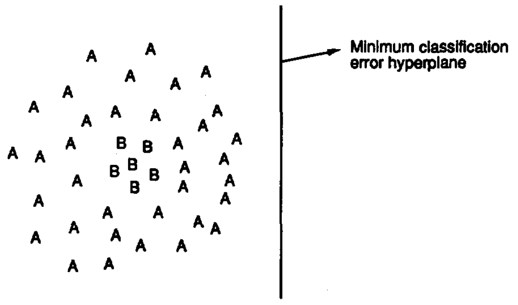


Fig. 3. The minimum classification error solution to the "donut" problem illustrated in this figure is to classify all the patterns as "A." Any attempt to get even one "B" vector correctly classified will result in a poorer solution in terms of minimum classification error, since many "A" vectors will then be incorrectly classified.

practice. However, this problem can be corrected by retraining the neuron using the training vectors from each class which resulted in maximum error. The resulting hyperplane is then guaranteed to split the data, since it is always possible to separate two opposite class vectors.

C. Generalization to Multiclass Problems

For an M-class problem, M neurons are used in the root node of the NTN. The classes are labeled using the M binary basis vectors. Thus class i is labeled with a 1 in the ith bit, all other bits being 0. This is called the local encoding scheme. After training the neural network, the feature vectors are classified according to a *winner take all rule* so that

$$\boldsymbol{x} \in \text{class } i \quad \text{iff } f(\boldsymbol{w}_i \cdot \boldsymbol{x}) \geq f(\boldsymbol{w}_j \cdot \boldsymbol{x}) \quad \forall j \neq i \qquad (7)$$

where f is the sigmoid activation function and $\boldsymbol{w}_i$ is the weight vector of the ith neuron. The winner take all rule divides the feature space into M convex regions [18]. The training data in each region is assigned to a different child node. Thus there are at most M child nodes. There may be less than M child nodes, since some of the M regions may not contain training data as in the "donut" problem illustrated previously in Fig. 3. In that problem, one of the two regions was empty. For each child node, the number of classes is found and the algorithm is repeated. As in the two-class problem, the neural network may not split the training data. However, if 100% classification on the training set is required, the neural network can be retrained using the training vector from each class that gives maximum error, guaranteeing a split. In our simulations we ignored this condition and the NTN still correctly classified more than 99% of the training data. Since the number of classes cannot increase, the number of neurons in the child nodes is equal to or less than M. Clearly the worst case is that every NTN node contains M neurons. However, in practice, this number decreases from the root to the leaves. Fig. 4 shows an NTN and the corresponding division of feature space for a 3-class pattern recognition problem. The neural network at the root divides the feature space into 3 regions corresponding to 3 child nodes. One of these regions has only one class in it and, thus, the corresponding child node is a leaf node. The other

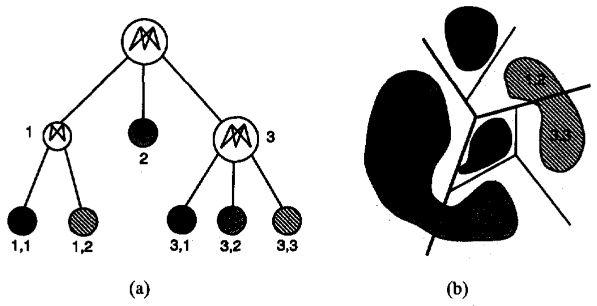


Fig. 4. (a) Two level NTN with neural nets in internal nodes. Leaf nodes shaded according to appropriate class. (b) Feature space regions: first number refers to first level split of feature space, and second number to a second level split. First level region boundaries are shown by thick lines.

two regions are further divided into subregions by the neural networks at the child nodes.

III. Pruning the NTN

The learning algorithm described in Section II-B will grow an NTN that performs well on the training data. This is expected since the algorithm keeps dividing the feature space until the training data is correctly classified. However, the performance on an *independent test set* will usually be poorer than the performance on the training set. The performance on the test set can be even poorer if the number of training data is small. These properties are true for any classification system. For this reason, it is important to form an estimate of the performance of the NTN that will hold up against an independent test set. The solution that has been suggested in the case of decision trees is to grow a large tree and then prune the tree [4]. This approach can also be used for the NTN [20]. Thus, the NTN grown by the learning algorithm is pruned to get the best performance on the test set.

Before proceeding further, some notation and definitions should be established. Fig. 5 illustrates these definitions. Let the NTN grown by the learning algorithm be denoted by T.

Definition 1: A trivial tree consists of just a single node.

Definition 2: A subtree T' of T has the same root node as T and if $t \in T'$, then $t \in T$. This relation is written as $T' \leq T$. If $T' \neq T$, then $T' < T$.

Definition 3: A branch T_t of T is defined as the union of the node t and all the descendants of t.

Definition 4: Let the root of T be denoted as t_1. The primary branches of T are the set of all branches rooted at the child nodes of t_1 and are denoted by T_i, $1 \leq i \leq N$, where N is the number of child nodes of t_1.

Definition 5: Pruning a branch T_t from T consists of deleting from T all the descendants of t. The new tree is now $T - T_t$. The node t now becomes a leaf node and is labeled by the class that has maximum membership in $\mathcal{X}_t$, the subset of training data corresponding to t. The pruned tree is called a pruned subtree of T.

The pruning algorithm generates a nested sequence of pruned subtrees so that each pruned subtree can be found from the previous pruned subtree. This allows for a simple and elegant pruning algorithm. The pruning algorithm makes use of a Lagrangian cost function given by

$$C_\alpha(T) = D_T + \alpha|\tilde{T}| \tag{8}$$

where D_T is the number of misclassifications of the NTN, T, on the training data. $|\tilde{T}|$ is the number of leaf nodes of T, and α is a penalty associated with each leaf node. The cost function thus takes into consideration the performance of the NTN and also its complexity, measured by the number of leaf nodes. If $\alpha = 0$, then the best pruned NTN is T. On the other hand, if $\alpha = \infty$, then the best pruned NTN is the root of T denoted by $\{t_1\}$. As α is increased from 0 to ∞, a sequence of optimally pruned subtrees is generated. We now define an optimally pruned subtree with respect to the cost function of (8).

Definition 6: A pruned subtree T_1 of T is an optimally pruned subtree with respect to α if $C_\alpha(T_1) = \min_{T' \leq T} C_\alpha(T')$. T_1 is the smallest optimally pruned subtree of T with respect to α if $T_1 \leq T'$ for all the optimally pruned subtrees, T' of T. There can be at most one smallest optimally pruned subtree of T. This subtree is denoted as $T(\alpha)$.

The pruning algorithm works by finding the sequence of smallest optimally pruned subtrees of T as α ranges from 0 to ∞. This sequence starts with T and ends with the root node $\{t_1\}$. Even though α takes on a continuum of values, there are only a finite number of subtrees of T. Thus T remains the smallest optimally pruned subtree for a range of values of $\alpha, 0 \leq \alpha < \alpha_1$. At this point a new subtree $T(\alpha_1)$ becomes the smallest optimally pruned subtree and continues to remain so until $\alpha = \alpha_2$. Now $T(\alpha_2)$ becomes the smallest optimally pruned subtree. This process continues until at some value of $\alpha = \alpha_L$, the root $\{t_1\}$ remains. $\{t_1\}$ then continues to be the smallest optimally pruned subtree for $\alpha \geq \alpha_L$.

We now present the pruning algorithm. Consider a node $t \in T$ and the branch, T_t. Using the cost function of (8), the cost of node t can be written as

$$C_\alpha(t) = D_t + \alpha \tag{9}$$

where D_t is the number of misclassifications of the node t. Note that the trivial tree t has only one leaf. The cost of the branch T_t is

$$C_\alpha(T_t) = D_{T_t} + \alpha|\tilde{T}_t| \tag{10}$$

where D_{T_t} is the number of misclassifications of the branch T_t and $|\tilde{T}_t|$ is the number of leaf nodes of the branch T_t. As long as $C_\alpha(t) > C_\alpha(T_t)$, the branch T_t is not pruned off. As α is increased, there is a point when $C_\alpha(t) = C_\alpha(T_t)$. At this point, T_t is pruned off since the resulting NTN, $T - T_t$, is less complex but has the same cost as T. This value of α is given by equating the right hand sides of (9) and (10). Solving for α, we get

$$\alpha = g(t, T) = \frac{D_{T_t} - D_t}{1 - |\tilde{T}_t|}. \tag{11}$$

The pruning algorithm finds this value of α for all the nodes $t \in T - \tilde{T}$ and then prunes off the branch T_{t^*} corresponding to the minimum value of $\alpha_t = \alpha_1$. The node t^* is now a leaf node and is labeled by the class with maximum membership

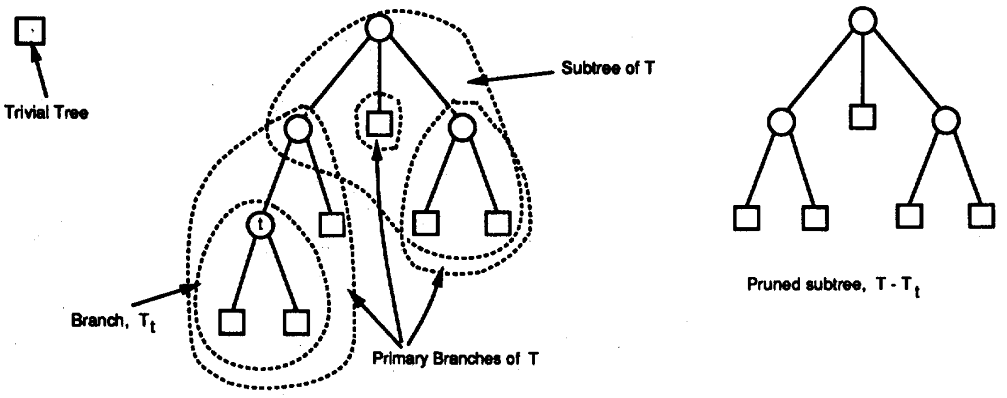


Fig. 5. Tree definitions.

in the training data in t^*. This process is repeated starting with the NTN, $T_1 = T - T_{t^*}$ and so on until only the root $\{t_1\}$ is left. Thus the algorithm produces a sequence of NTN's $T, T_1, T_2 \cdots \{t_1\}$.

Suppose, at each stage i of the algorithm, there are N_i nodes in the subtree T_i. Then, $O(N_i)$ ADD and MULTIPLY operations are needed to compute α for each node of T_i and $O(N_i)$ COMPARE operations are required to find the minimum value of α. In the worst case, the number of pruned subtrees in the sequence generated by the algorithm is N_T, where N_T is the number of nodes in the NTN grown by the learning algorithm. Thus the number of operations required to compute the sequence of pruned subtrees is, in the worst case, given by $O(\sum_{i=1}^{N_T} i) = O(N_T^2)$. In practice, however, the complexity is smaller. The complexity of the pruning algorithm is also small compared to the training algorithm, since the number of training vectors and the number of iterations for training is usually much larger than N_T.

We now show that the sequence of subtrees generated by the pruning algorithm is the sequence of smallest optimally pruned subtrees, $T, T(\alpha_1), \cdots \{t_1\}$. The following theorem, which is used in the proof, is given first.

Theorem 1: If $\alpha_2 \geq \alpha_1$, then $T(\alpha_2) \leq T(\alpha_1)$.

Proof: The theorem is clearly true if T_1 is a trivial tree, i.e. if $T_1 = T(\alpha_1)$, then $T_1 = T(\alpha_2)$, for $\alpha_2 \geq \alpha_1$. Consider the tree T rooted at t_1 and assume the theorem is true for all primary branches T_i, $1 \leq i \leq N$, where N is the number of child nodes of t_1, i.e.,

$$\text{if } \alpha_2 \geq \alpha_1, \text{ then } T_i(\alpha_2) \leq T_i(\alpha_1), 1 \leq i \leq N. \quad (12)$$

The cost of T can be written in terms of the costs of T_i as

$$C_\alpha(T) = \sum_{i=1}^{N} C_\alpha(T_i). \quad (13)$$

Let $T_1 = T(\alpha)$. From (13) it can be shown by mathematical induction that

$$T_1 = \{t_1\}, \text{ if } C_\alpha(t_1) \leq \sum_{i=1}^{N} C_\alpha(T_i(\alpha)) \quad (14)$$

$$T_1 = \{t_1\} \cup (\cup_{i=1}^{N} T_i(\alpha)), \text{ if } C_\alpha(t_1) > \sum_{i=1}^{N} C_\alpha(T_i(\alpha)). \quad (15)$$

From this and the assumption of statement (12) above, Theorem 1 follows by mathematical induction

This theorem paves the way for the next theorem which proves the optimality of the pruning algorithm.

Theorem 2: The pruning algorithm given above generates the sequence of smallest optimally pruned subtrees $T > T(\alpha_1) > T(\alpha_2) \cdots > T(\alpha_{L-1}) > \{t_1\}$.

Proof: It is clear that T is the smallest optimally pruned subtree of itself with respect to α, for $0 \leq \alpha < \alpha_1$, and that $T_1 = T(\alpha_1)$ is the smallest optimally pruned subtree of T with respect to $\alpha_1 = \min_{t \in T - \tilde{T}} g(t, T)$, where $g(t, T)$ is defined by (11). Also T_1 is the smallest optimally pruned subtree of itself with respect to α, for $\alpha_1 \leq \alpha < \alpha_2$ and T_2 is the smallest optimally pruned subtree of T_1 with respect to $\alpha_2 = \min_{t \in T_1 - \tilde{T}_1} g(t, T_1)$, i.e., $T_2 = T_1(\alpha_2)$. We shall show that $\alpha_2 > \alpha_1$, $T(\alpha) = T_1$ for $\alpha_1 \leq \alpha < \alpha_2$, and that $T(\alpha_2) = T_2$.

From theorem 10.11 of [4], we know that if $t \in T_1 - \tilde{T}_1$, then

$$\begin{aligned} g(t, T_1) &> g(t, T) \quad \text{if } T_{1_t} < T_t \\ g(t, T_1) &= g(t, T) \quad \text{if } T_{1_t} = T_t. \end{aligned}$$

But $\alpha_2 = \min_{t \in T_1 - \tilde{T}_1} g(t, T_1)$, and $\alpha_1 = \min_{t \in T - \tilde{T}} g(t, T)$ and thus $\alpha_2 > \alpha_1$.

If $\alpha_1 \leq \alpha < \alpha_2$, then, from Theorem 1, $T(\alpha) \leq T(\alpha_1) = T_1 < T$. From this and the fact that the relation $\leq$ is transitive, we get $T(\alpha) = T_1(\alpha) = T_1$. Again from Theorem 1, $T(\alpha_2) \leq T(\alpha_1) = T_1 < T$ and from the transitivity of $\leq$, we get $T(\alpha_2) = T_1(\alpha_2) = T_2$. If T_2 is trivial, then $T(\alpha) = T_2$ for $\alpha \geq \alpha_2$, otherwise the process can be repeated until only the root $\{t_1\}$ remains. This proves Theorem 2.

After generating the sequence of smallest optimally pruned subtrees, an independent test set of data is used to evaluate each pruned subtree. The best subtree is picked and used for classification purposes. If the number of training and test data is large, this will give an unbiased estimate of the performance of the classifier [4].

IV. Numerical Results

The performance of the NTN is first demonstrated on two Boolean function learning problems. These are the exclusive OR (XOR) problem, and a four variable nonlinearly separable function given by $(a \vee b) \oplus (c \wedge d)$. We then present results on a speaker independent vowel recognition task. Both the NTN and MLP use neurons with sigmoid activation functions, resulting in slower learning when the output is close to the saturation point of the sigmoid. In our experiments the target outputs for the neurons were selected to be 0.1 and 0.9 for both the NTN and the MLP.

A. Boolean Functions

In these tasks, a function completely specified by the training data is to be learned by the system. Since all input-output pairs are present in the training data, there is no separate test set, and pruning is not used. The training speed is thus the speed of the NTN learning algorithm. The XOR function of two variables is given by $a \oplus b$. The Boolean vectors, $(0,0)$, and $(1,1)$ belong to one class and the vectors, $(0,1)$, and $(1,0)$ belong to the second class.

Fig. 6 shows the NTN grown for the XOR problem and also the MLP that solves the problem. For the NTN, the internal nodes are drawn by circles and the leaf nodes are drawn by squares. The leaf nodes are used only for classification. Processing using neurons takes place only in the internal nodes. The NTN uses only two processing nodes, each having one neuron, whereas the MLP needs three neurons. Table II tabulates the number of epochs taken by the NTN algorithm and backpropagation to learn the XOR problem. One epoch for the backpropagation algorithm consists of one presentation of all the training data to the network. Thus, in the XOR problem, which has 4 training vectors, each weight of the MLP is updated 4 times in each epoch. In the NTN, an epoch is defined on a node by node basis. Thus, one epoch for node t is one presentation of the training data in node t to the neural network in node t. Since the number of training data decreases from the root to the leaf nodes, the NTN epoch becomes less costly as we go down the NTN. Also, in each NTN epoch, the weights of a small neural network (in this case, a single neuron) in a particular node of the NTN are updated rather than the weights of a large neural network as in the MLP. The number of NTN epochs in Table II is the sum of the epochs for all the nodes of the NTN. Since an NTN epoch has fewer weight updates than an MLP epoch, we also compare the total number of weight updates required for each algorithm. As seen in the table, the NTN requires 30 times fewer weight updates. The computational complexity of a weight update is the same for both the MLP and the NTN. Thus, we conclude that the NTN trains 30 times faster than the MLP for this problem. It is theoretically possible for an MLP to adapt all its weights in parallel whereas an NTN exhibits parallelism only within the weight updates in the same level of the tree. Taking this into consideration, the NTN was found to train 10 times faster than the MLP.

We now present results on a nonlinearly separable four variable Boolean function given by $(a \vee b) \oplus (c \wedge d)$. For

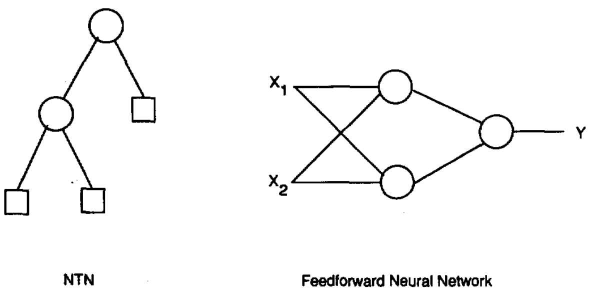


Fig. 6. NTN and MLP for XOR

TABLE II
Number of Learning Epochs to Solve XOR

Algorithm	Number of Epochs	Number of Weight Updates
NTN	30	315
Backpropagation	256	9216

this problem, we compare the performance of the NTN with backpropagation and also two decision tree approaches. The first, ID3 [21], is a popular decision tree method used by the Artificial Intelligence community. ID3 splits the training data at each internal node on the basis of the value of a feature variable selected by an exhaustive search over all variables. The second, Perceptron Trees [22], uses ID3 nodes for the internal tree nodes and perceptrons at the leaf nodes. Thus, if the training set at a tree node is not linearly separable, then an ID3 node is used and if the set is linearly separable, a perceptron is used. Since the NTN uses a neuron at all tree nodes, its performance is better than these methods.

Fig. 7 shows the MLP that solves this problem and also the trees which were grown using the NTN, ID3, and perceptron trees. The NTN requires only 3 internal nodes as compared to 8 nodes for ID3 and 5 nodes for the perceptron trees. The ID3 and perceptron tree results were taken from [22]. The MLP required 3 hidden neurons for this problem.

Table III compares the NTN learning algorithm with backpropagation in terms of the number of epochs and the number of weight updates needed to learn the problem. As can be seen from the number of weight updates, the new algorithm trained about 50 times faster than backpropagation. Again, assuming that the weights are updated in parallel, the NTN was found to be about 12 times as fast as the MLP.

B. Speaker Independent Vowel Recognition

Speech recognition is a problem that has received considerable attention in the past [23], [24]. Recently there has been much work on the use of neural networks for speech recognition [7], [25]–[27]. Comparative studies of neural networks and decision trees for speech recognition have also been performed [28], [29].

In this section, we present simulation results on a speaker independent vowel recognition task. The database for this problem is maintained as a benchmark by Scott Fahlman at CMU and there have been results published for this task using MLP's [30] and decision trees [28]. The training set consists

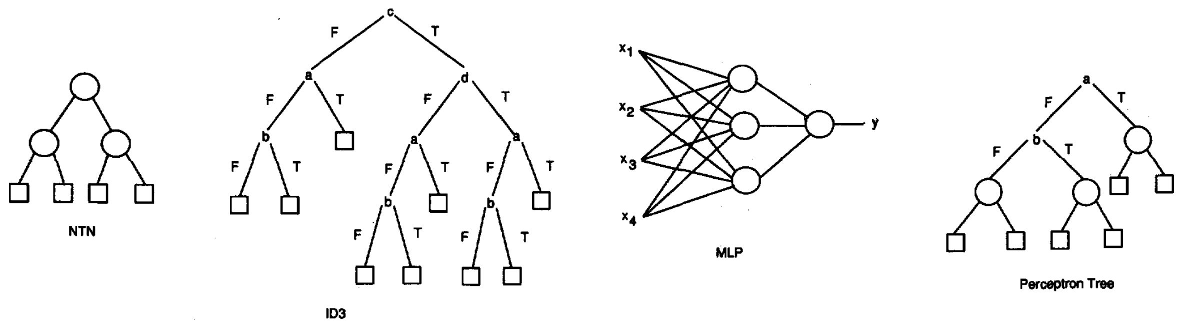


Fig. 7. Different classifiers for $(a \vee b) \oplus (c \wedge d)$.

TABLE III
NUMBER OF LEARNING EPOCHS TO SOLVE $(a \vee b) \oplus (c \wedge d)$

Algorithm	Number of Epochs	Number of Weight Updates
NTN	77	4350
Backpropagation	715	217360

of 528 feature vectors representing 11 vowel sounds obtained from 4 male speakers and 4 female speakers. The problem is to train the system on this data and then test it on an independent set of 462 feature vectors obtained from a different set of 4 male speakers and 3 female speakers. For both the training and testing set, each vowel is repeated 6 times by each speaker. The features are 10 log area parameters calculated using six 512 sample Hamming windowed segments from the steady part of the vowel. There is no context information provided to the classification system. Details of this data set can be obtained from [30]. This particular database represents a difficult problem and the best performance of any classifier was only around 55%.

Experiments have been performed on this data set using various MLP neural networks with different numbers of hidden nodes and different types of nodes, such as radial basis function nodes and Gaussian nodes [30]. The performance of CART on this data set has been reported in [28]. These results are summarized in Table IV. The table also shows the performance of the best pruned NTN on this data set. These results have been averaged over 20 runs of the NTN algorithm. The performance of the NTN is much better than CART but is similar in classification accuracy to MLP's with radial basis function (RBF) nodes, Gaussian nodes (GN) or square nodes (SN). Due to pruning, the performance of the NTN on the training set decreased from almost 100% to 84%. This decrease is expected but the resulting NTN gives the best performance on the test set (see Section III). As shown in the table, if the number of hidden nodes is not chosen correctly, the performance of MLP's is quite poor. The NTN algorithm, however, grows the correct number of neurons while training.

Table IV also shows the number of neurons used in the various methods. The number of neurons used in the NTN is less than that of MLP's with similar performance. The average depth of the NTN was only 3.1, which is quite a shallow tree. The average classification time for a test pattern is, thus, only 3.1 time units. An MLP with one hidden layer and implemented in parallel requires two time units—one to

TABLE IV
RESULTS ON SPEAKER INDEPENDENT VOWEL RECOGNITION

Classifier	Number of Hidden Units	Percent Correct
Single-Layer Perceptron	–	33
MLP	88	51
MLP	22	45
MLP	11	44
RBF	528	53
RBF	88	48
GN	528	55
GN	88	53
GN	22	54
GN	11	47
SN	88	55
SN	22	51
SN	11	50
CART	–	44
NTN	59[1]	54

[1]This is the total number of neurons used in the NTN.

fire the hidden layer nodes and the second for the output nodes. The NTN is therefore only very slightly slower than the MLP for retrieval. However, only the 11 neurons in the root node need to be implemented for the NTN. These neurons can be re-used as mentioned in Section II-A for the two-class case.

V. SUMMARY AND CONCLUSIONS

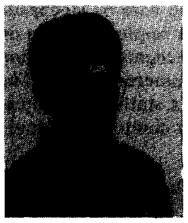
We have introduced a new classification method called Neural Tree Networks which combines decision trees and neural networks. The NTN offers an attractive implementation tradeoff over MLP's. The new learning algorithm grows the network as opposed to backpropagation where the MLP architecture has to specified before learning can begin. The new algorithm forms splits of the training data using a gradient descent technique and does not have to resort to exhaustive search techniques as used in decision trees. An optimal pruning algorithm was given by generalizing the pruning algorithm of [4] for binary classification trees. Simulation results show that the new method grows smaller trees than conventional decision tree approaches and also trains faster than backpropagation. The NTN classification performance is similar to that of the best MLP's and is achieved without having to use trial and error techniques to generate the appropriate architecture. The NTN was also found to have fewer neurons than an MLP with similar performance.

ACKNOWLEDGMENT

This research was performed in partial fulfillment of the requirements for the Ph.D. degree of Ananth Sankar

REFERENCES

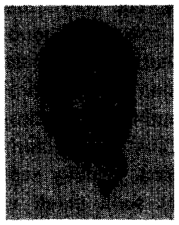
[1] D. Rumelhart and J. McClelland, *Parallel Distributed Processing.* Cambridge, MA: M.I.T. Cambridge Press, 1986.
[2] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., 1974.
[3] D. Parker, "Learning logic," Tech. Rep. TR-47, M.I.T., Center for Computational Research in Economics and Management Science, 1985.
[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Belmont, CA: Wadsworth International group, 1984.
[5] R. Duda and P. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.
[6] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* New York: Academic, 1972.
[7] A. Waibel, "Modular construction of time delay neural networks for speech recognition," *Neural Computation,* vol. 1, Mar. 1989.
[8] A. Rajavelu, M. Musavi, and M. Shirvaikar, "A neural network approach to character recognition," *Neural Networks,* vol. 2, no. 5, pp. 387–394, 1989.
[9] Y. L. Cun, O. Matan, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, and H. Baird, "Handwritten zip code recognition with multilayer networks," in *Proc. 10th Int. Conf. Pattern Recognition,* vol. 2, 1990, pp. 35–40.
[10] S. Judd, "Learning in networks is hard," in *Proc. IEEE First Int. Conf. Neural Networks,* vol. 2, June 1987, pp. 685–692.
[11] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.,* 1965.
[12] E. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity,* vol. 4, pp. 193–215, Sept. 1988.
[13] L. Hyafil and R. Rivest, "Constructing optimal decision trees is NP-complete," *Inform. Processing Lett.,* vol. 5, no. 1, pp. 15–17, 1976.
[14] S. Amari, "A theory of adaptive pattern classifiers," *IEEE Trans. Electron. Comput.,* vol. EC-16, pp. 299–307, June 1967.
[15] A. Sankar and R. Mammone, "A new fast learning algorithm for feedforward neural networks using the L1 norm of the error," Tech. Rep. CAIP TR-115, CAIP Center, Rutgers Univ., 1990.
[16] M. Frean, "Small nets and short paths: Optimising neural computation," Ph.D. dissertation, Univ. Edinburgh, 1990.
[17] B. Juang, personal communication.
[18] A. Sankar and R. Mammone, "Neural tree networks," in *Neural Networks: Theory and Applications,* R. Mammone and Y. Zeevi, Eds. New York: Academic, 1991, pp. 281–302.
[19] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science,* vol. 220, May 1983.
[20] A. Sankar and R. Mammone, "Optimal pruning of neural tree networks for improved generalization," in *Proc. IJCNN,* July 1991, pp. II-219–II-224,.
[21] J. Quinlan, "Induction of decision trees," *Machine Learning,* vol. 1, pp. 81–106, 1986.
[22] P. Utgoff, "Perceptron trees: A case study in hybrid concept representation," in *Proc. Seventh Nat. Conf. Artif. Intell.,* St. Paul, MN, Morgan-Kaufman, 1988.
[23] J. Flanagan, *Speech Analysis Synthesis and Perception.* Berlin, Germany: Springer-Verlag, 1972.
[24] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals.* Englewood Cliffs, NJ: Prentice-Hall, 1978.
[25] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural Computation,* vol. 1, no. 1, pp. 1–38, 1989.
[26] K. Unnikrishnan, J. Hopfield, and D. Tank, "Connected-digit speaker-dependent speech recognition using a neural network with time-delayed connections," *IEEE Trans. Signal Processing,* vol. 39, pp. 698–713, Mar. 1991.
[27] K. Unnikrishnan, J. Hopfield, and D. Tank, "Speaker-independent digit recognition using a neural network with time-delayed connections," *Neural Computation,* vol. 4, no. 1, 1991.
[28] A. C. Tsoi and R. Pearson, "Comparison of three classification techniques, CART, C4.5, and multi-layer perceptrons," presented at the NIPS Post Conference Workshop, Denver, CO, 1990.
[29] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawi, and R. M. II, "A performance comparison of trained multilayer perceptrons and trained classification trees," *Proc. IEEE,* vol. 78, Oct. 1990.
[30] A. Robinson, "Dynamic error propagation networks," Ph.D. dissertation, Cambridge Univ. Eng. Dep., 1989.



Ananth Sankar (S'90–M'91) received the B.Tech. degree from the Indian Institute of Technology, Kanpur, in 1986 and the M.S. degree from the State University of New York, Stony Brook, in 1987 both in electrical engineering and the Ph.D. degree from Rutgers University, Piscataway, NJ, in 1991, in electrical and computer engineering.

From October, 1987 to August, 1988, he was a lecturer at the Department of Computer Science, University of Poona, Pune, India. He is currently with the Information Principles Research Laboratory, AT&T Bell Laboratories, Murray Hill, NJ. His research interests are in pattern recognition and classification, signal and image processing, neural networks, language acquisition, and learning systems.

Dr. Sankar is a member of Eta Kappa Nu and Sigma Xi.



Richard J. Mammone (S'75–M'81–SM'86) received the B.E.E., M.E.E., and the Ph.D. degrees from the City University of New York in 1975, 1977, and 1981, respectively.

He is currently an Associate Professor of electrical and computer engineering at Rutgers University, Piscataway, NJ. His research and teaching interests are in the areas of image and speech processing and neural networks. He has numerous publications and patents in these areas and is a frequent consultant to industry and government agencies.

Dr. Mammone is an Associate Editor of the journal *Pattern Recognition* and was an Associate Editor of IEEE COMMUNICATIONS magazine. He is a co-editor of a book on neural networks with Y.Y. Zeevi. He is a member of OSA, SPIE, Eta Kappa Nu, and Sigma Xi.