5 Summary and Conclusions

In this paper, we extended the optimal pruning algorithm for binary classification trees [7] to arbitrary trees such as the NTN. The pruning algorithm was used to improve the generalization of the NTN. We presented simulation results on a speaker independent vowel recognition task and compared the performance of the NTN with MLPs and decision trees. The NTN was shown to be superior to both these methods. We are generalizing the pruning algorithm to other constructive neural network algorithms such as the Upstart Algorithm [10] and the algorithm given in [11]

Acknowledgments

The research reported here was made possible through the support of the New Jersey Commission on Science and Technology and the Center for Computer Aids for Industrial Productivity at Rutgers University.

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

References

[1] E. Baum, "What Size Net Gives Valid Generalization?," *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989.

[2] D. Rumelhart and J. McClelland, *Parallel Distributed Processing*. MIT Cambridge Press, 1986.

[3] M. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," in *Advances in Neural Information Processing I* (D. Touretzky, ed.), pp. 107–115, Morgan-Kaufmann, 1989.

[4] E. D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 239–242, June 1990.

[5] A. Sankar and R. Mammone, "A Fast Learning Algorithm for Tree Neural Networks," in *Proceedings of the 1990 Conference on Information Sciences and Systems*, pp. 638–642, March 1990.

[6] A. Sankar and R. Mammone, "Neural Tree Networks," in *Neural Networks: Theory and Applications* (R. Mammone and Y. Zeevi, eds.), Academic Press, 1991.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Belmont,CA: Wadsworth international group, 1984.

[8] A. Robinson, *Dynamic Error Propagation Networks*. PhD thesis, CAmbridge University Engineering Department, 1989.

[9] A. C. Tsoi and R. Pearson, "Comparison of Three Classification Techniques, CART, C4.5, and Multi-Layer Perceptrons." Proceedings of the 1990 IEEE Conference on Neural Information Processing Systems, Denver.

[10] M. Frean, "The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks," *Neural Computation*, pp. 199–209, 1990.

[11] J.-P. Nadal, "Study of a Growth Algorithm for a Feedforward Network," *International Journal of Neural Systems*, vol. 1, no. 1, pp. 55–59, 1989.

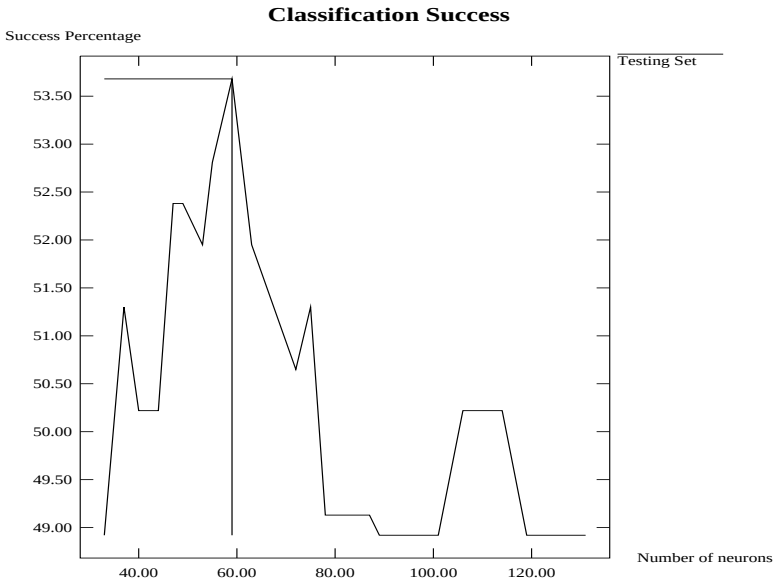

Figure 1: Classification error for the sequence of optimally pruned subtrees: X-axis: Number of neurons: pruning proceeds from right to left, Y-axis: Classification success percentage

Classifier	MLP	MLP	MLP	RBF	RBF	GN	GN	CART	NTN
Number of hidden neurons	88	22	11	528	88	528	88	–	59 [2]
Percent Correct	51	45	44	53	48	55	53	44	54

Table 1: Results on Speaker Independent Vowel Recognition,MLP: Multi-Layer Perceptron, RBF: Radial Basis Function, GN: Gaussian Nodes, CART: A Decision Tree Approach, NTN: Neural Tree Network

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.*

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

After finding the sequence of pruned NTNs, an independent test set of examples is run through each pruned NTN, $T(\alpha_i)$, to find the NTN with the smallest classification error. If the number of test vectors is large, this provides a relatively unbiased estimate of the classification error of the NTN [7].

4 Simulation Results

In this section, we present simulation results on a speaker independent vowel recognition task. The training set consists 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 state part of the vowel. Details of this data set can be obtained from [8].

The NTN learning algorithm was first used to grow the NTN.The pruning algorithm was then used to generate a sequence of optimally pruned subtrees. Finally the testing set was used to find the classification error for each of the pruned subtrees. In the graph of figure 1, we show a plot of the classification success percentage for the testing set for each of the pruned subtrees. The success rate is plotted against the number of neurons used in the pruned subtrees. Since the number of neurons decreases while pruning, the X-axis of the graph should be read from right to left. In this particular run of the algorithm there are 30 pruned subtrees. The graph shows the success rate for the first 27 optimally pruned subtrees. As can be seen from the graph, the classification success rate increases to a maximum and then decreases again as pruning is continued. In this case the pruned subtree with the minimum error on the testing set is $T(\alpha_{17})$. The number of neurons used in this NTN was 59.

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 guassian nodes [8]. The performance of CART [7], one of the most popular decision tree methods, on this data set has been reported in [9]. These results are summarized in table 1. 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. We see that the performance of the NTN is much better than CART but is similar in classification accuracy to MLPs with radial basis function (RBF) nodes or guassian nodes (GN). As shown in the table, if the number of hidden nodes is not chosen correctly, the performance of MLPs is quite poor. The NTN algorithm, however, grows the correct number of neurons while training. Table 1 also shows the number of neurons used in the various methods. The number of neurons used in the NTN is significantly less than that of MLPs with similar performance. The average maximum depth of the NTN was only 3.1 with a worst case maximum depth of 4, which is quite a shallow tree. The retrieval time for a test pattern is thus only around 3 to 4 time units. An MLP with one hidden layer and implemented in parallel requires two time units–one to fire the hidden layer nodes and the second for the output nodes. The NTN is therefore only slightly slower than the MLP for retrieval. However, for the NTN, only one processor needs to be implemented [6], whereas all the processors must be implemented in the case of the MLP. Thus the NTN provides an attractive trade off of speed against hardware complexity as compared to MLPs.

[2] The neurons in the NTN are not hidden neurons. Rather this number refers to the total number of neurons used in the NTN

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 $\alpha \geq \alpha_L$. Pruning consists of finding the sequence of subtrees $T, T(\alpha_1), T(\alpha_2), \ldots, T(\alpha_{L-1}), \{t_1\}$, for $0 < \alpha_1 < \alpha_2 \ldots < \alpha_{L-1} < \alpha_L$.

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.*

$$\text{if } \alpha_2 \geq \alpha_1, \text{then } T_i(\alpha_2) \leq T_i(\alpha_1), 1 \leq i \leq N \tag{3}$$

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

$$C_\alpha(T) = \sum_{i=1}^{N} C_\alpha(T_i) \tag{4}$$

Let $T_1 = T(\alpha)$. From equation 4 it can be shown by mathematical induction that

$$T_1 = \{t_1\}, \quad \text{if } C_\alpha(t_1) \leq \textstyle\sum_{i=1}^{N} C_\alpha(T_i(\alpha)) \tag{5}$$
$$T_1 = \{t_1\} \cup (\cup_{i=1}^{N} T_i(\alpha)), \quad \text{if } C_\alpha(t_1) > \textstyle\sum_{i=1}^{N} C_\alpha(T_i(\alpha)) \tag{6}$$

From this and the assumption of statement 3 above, the theorem follows by mathematical induction

From theorem 1, it is clear that if $\alpha_2 > \alpha_1$ and $T(\alpha_2) \neq T(\alpha_1)$, then $T(\alpha_2) < T(\alpha_1)$. Thus the sequence of smallest optimally pruned subtrees is a nested sequence, $T > T(\alpha_1) > T(\alpha_2) \ldots > T(\alpha_{L-1}) > \{t_1\}$. We now describe the algorithm to find this sequence of pruned subtrees.

Algorithm
The full NTN, T, is grown by the NTN learning algorithm. For all nodes $t \in T - \tilde{T}$, if $C_\alpha(t) > C_\alpha(T_t)$, then the branch, T_t, is preferable to the node t and is kept in the tree. If $C_\alpha(t) = C_\alpha(T_t)$, then t is preferable to T_t since it has fewer leaf nodes and is thus less complex. This value of α is given by:

$$\alpha = g(t, T) = \frac{D_{T_t} - D_t}{1 - |\tilde{T}_t|} \tag{7}$$

The branch, T_{t^*}, corresponding to $\alpha_1 = g(t^*, T) = \min_{t \in T - \tilde{T}} g(t, T)$ is pruned to get $T_1 = T - T_{t^*}$. We now start with the tree T_1 and repeat the above procedure to find $T_2 = T_1 - T_{1_{t^{**}}}$ corresponding to $\alpha_2 = g(t^{**}, T_1) = \min_{t \in T_1 - \tilde{T}_1} g(t, T_1)$. The algorithm is again repeated starting with T_2 until, ultimately, only the root $\{t_1\}$ remains. $g(t, T)$ for all the nodes $t \in T$ can be calculated and stored at each node t while the NTN is being grown. The minimum value of $g(t, T)$ can then be efficiently found by using a recursive search technique that exploits the tree structure.

Theorem 2 *The algorithm given above generates the sequence of smallest optimally pruned subtrees $T > T(\alpha_1) > T(\alpha_2) \ldots > T(\alpha_{L-1}) > \{t_1\}$.*
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 $\alpha_1 = \min_{t \in T - \tilde{T}} g(t, T)$. 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 $\alpha_2 = \min_{t \in T_1 - \tilde{T}_1} g(t, T_1)$. 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 [7], we know that if $t \in T_1 - \tilde{T}_1$, then

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

leaf nodes of the NTN. The leaf nodes are labeled according to the class they represent. The training set is repeatedly divided so as to decrease the classification error at each step [5, 6]. The NTN classifies an input feature vector by using the neural network at the internal NTN nodes to route the feature vector from the root node to the appropriate leaf node.

Let us denote the NTN grown by the learning algorithm by T and the set of leaf nodes as $\tilde{T}$. The classification error of the NTN is given by

$$D_T = \sum_{t \in \tilde{T}} d_t p_t \tag{1}$$

where d_t is the number of misclassifications at node t and p_t is the probability that a training vector gets routed to node t. The classification error at node t is thus $D_t = d_t p_t$. The learning algorithm decreases D_T at each step. In the context of pruning, the classification error $D_{T'}$ on the training set for a subtree T' of T is always greater than D_T. Hence pruning always increases the number of misclassifications on the training set. The learning algorithm grows the NTN so that D_T is below some small tolerance level. However, for good generalization, it is necessary to prune the NTN as described in the next section.

3 The Pruning Algorithm

An optimal pruning algorithm for binary classification trees has been given in [7]. We extend this algorithm to the case of arbitrary trees such as the NTN. We first establish some notation and definitions. Denote the NTN grown by the learning algorithm as 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 $T - T_t$. t now becomes a leaf node and is labeled by the class that has maximum membership in X_t, the subset of training data corresponding to t. The pruned tree is called a pruned subtree of T.*

The pruning algorithm makes use of the following Lagrangian cost function:

$$C_\alpha(T) = D_T + \alpha \mid \tilde{T} \mid \tag{2}$$

where α is the penalty attached to each leaf node and $\mid \tilde{T} \mid$ is the number of leaf nodes of T. $\mid \tilde{T} \mid$ is a measure of the complexity of the tree. This cost function reflects the trade off between the classification error of T and its complexity.

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 a sequence of smallest optimally pruned subtrees of T for all values of α from 0 to ∞. When $\alpha = 0$, the smallest optimally pruned subtree is T itself, and when $\alpha = \infty$, the smallest optimally pruned subtree is the root of T denoted by $\{t_1\}$. There are only a finite number of subtrees of T, but α takes on a continuum of values. 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

Optimal Pruning of Neural Tree Networks for Improved Generalization [1]

Ananth Sankar and Richard J. Mammone
CAIP Center and Dept. of Electrical Engineering
Rutgers University
P.O. Box 1390
Piscataway, NJ 08855-1390

Abstract

An optimal pruning algorithm for a Neural Network recently developed called Neural Tree Networks (NTN) is presented. The NTN is grown by a constructive learning algorithm that decreases the classification error on the training data recursively. The optimal pruning algorithm is then used to improve generalization. The pruning algorithm is shown to be computationally inexpensive. Simulation results on a speaker independent vowel recognition task are presented to show the improved generalization using the pruning algorithm.

1 Introduction

Obtaining the optimum size neural network is important for good generalization. If the network is very large, it tends to memorize the training data and thus generalize poorly. The size of the network depends on the number of training data and the degree of generalization required. Recently a relation has been given relating the size of an MLP, the training set size and the degree of generalization required [1]. Thus, it is possible to find the appropriate number of neurons for an MLP that will generalize well. However, there is no guarantee that learning algorithms such as backpropagation [2] will find the correct weights for a given number of neurons. A standard approach to get around this problem is to start with a large network and then to remove neurons that have little effect on the classification error [3, 4]. The neurons are ordered according to the amount by which the classification error is decreased when the neuron is removed. The neurons with the smallest effect are removed or pruned. These methods are not optimal in that for a given classification error, a smaller network than that found by these algorithms may exist.

In this paper, we present an optimal pruning algorithm for a new kind of neural network called Neural Tree Network (NTN) [5, 6]. This network has a tree structure with a neural network at each tree node. Unlike many standard classification trees, the NTN is not a binary tree. An optimal pruning algorithm for binary classification trees has been developed in [7]. We extend this approach to the case of arbitrary tree structures such as the NTN.

In section 2, the NTN is reviewed. The pruning algorithm is described in section 3. In section 4, simulation results on a speaker independent vowel recognition task are presented. The summary and conclusions are given in section 5.

2 The Neural Tree Network (NTN)

Recently a new neural architecture and learning algorithm called Neural Tree Network have been developed [5, 6]. The essential idea is to use a tree structure with a feedforward neural network with no hidden neurons at each internal (non-leaf) tree node. At the root of the NTN, the corresponding neural network is used to divide the training set, X, into at most N subsets, where N is the total number of classes. Each subset of X is assigned to a separate child node. This process is repeated for each child node in a recursive manner until each subset corresponds to a single class. These single class subsets correspond to

[1] To appear in the Proceedings of IJCNN, Seattle, 1991