Generating Neural Networks Through the Induction of Threshold Logic Unit Trees (Extended Abstract)

Mehran Sahami

Computer Science Department, Stanford University, Stanford, CA 94305, USA
Email: sahami@CS.Stanford.EDU

Abstract. We investigate the generation of neural networks through the induction of binary trees of threshold logic units (TLUs). Initially, we describe the framework for our tree construction algorithm and how such trees can be transformed into an isomorphic neural network topology. Several methods for learning the linear discriminant functions at each node of the tree structure are examined and shown to produce accuracy results that are comparable to classical information theoretic methods for constructing decision trees (which use single feature tests at each node). Our TLU trees, however, are smaller and thus easier to understand. Moreover, we show that it is possible to simultaneously learn both the topology and weight settings of a neural network simply using the training data set that we are given.

1 Introduction

We present a non-incremental algorithm that learns binary classification tasks by producing decision trees of threshold logic units (TLU trees). While similar to the decision trees produced by algorithms such as ID3 [QU, 1986], TLU trees promise more generality as each node in our tree implements a linear discriminant function as opposed to testing only one feature of the instance vector. Thus, if the data to be classified does not align closely with the principle axes of the instance space, a large number of single feature tests may be required to properly separate the data set, while just a few *oblique* separating functions could perform just as well. Such *multivariate* decision trees have only very recently begun to attract the attention of researchers in the machine learning community [BU, 1992; 1994].

Furthermore, we show how any such TLU trees can be mechanically transformed into a three-layer neural network as first suggested by Brent [BR, 1990] and developed by Sahami [SA, 1993]. In our investigation, we compare several different methods for learning the linear discriminant at each node of the tree and compare these with ID3's univariate approach and a naive Bayesian method for learning multivariate tests.

2 The TLU Tree Algorithm

The tree building algorithm is non-incremental requiring that the set of all training instances, S, be available from the outset. We begin with the root node of the tree and induce a hyperplane to separate the training set into the sets S_0 and S_1, where S_i ($i = 0, 1$) indicates the set of instances classified as i by the separating hyperplane. If S_0 contains instances labeled 1 we create a left child and recursively apply the algorithm to it using S_0 as the training set. Similarly, if S_1 contains instances labeled 0 we create a right child and again recursively apply our algorithm to it using S_1 as the training set. Thus the algorithm normally terminates when all of the instances in the original training set, S, are correctly classified by the tree. In our experiments, we stop growing a branch if a child produces no better split than its parent or the number of errors at the leaf is less than some prespecified level of error, E, to prevent overly complex trees — an application of Occam's Razor.

3 Creating Networks From TLU Trees

The trees which are produced by the TLU tree algorithm can be mechanically transformed into three-layer connectionist networks that implement the same functions. Given an TLU tree, T, with m nodes we can construct an isomorphic network containing the m nodes of the tree in the first hidden layer (each fully connected to the inputs). The second hidden layer consisting of a node (*AND* gate) for each possible *distinct* path between the root of T and a fringe node (any node without two children). Finally, the output layer is merely a single *OR* gate connected to all nodes in the previous layer. The connections between the first and second hidden layers are constructed by traversing each possible path from the root to a fringe node in T, and recording which nodes lie along such paths. Thus, each node in the second hidden layer represents a single distinct path through T by being connected to those nodes in the first layer which correspond to the nodes that were traversed along the given path. Since the nodes in the second hidden layer are merely *AND* gates, the inputs coming from the first hidden layer must first be inverted if a left branch was traversed at the node corresponding to a given input from the first hidden layer.

4 Experimental Results

4.1 Experimental Considerations

In our experiments we compare a number of methods for partitioning the instance space at each node of the tree. We consider our own variation of the ID3 algorithm in which single feature tests are selected based on minimizing entropy. The second method uses a naive Bayesian classifier to find an optimal separating hyperplane that minimizes the probability of error at each node *assuming the features of the instance are statistically independent.* Comparatively, we examine a number of adaptive techniques for finding hyperplanes: (i) the Perceptron error-correction rule [NI, 1965] employing the Pocket algorithm [GA, 1986], (ii) the Least Mean Square (LMS) algorithm [WW, 1988] with an annealled learning rate, and (iii) Back-propagation [RHW, 1986] applied to one neuron and then hard-thresholded *after* learning.

4.2 Learning Simple Boolean Functions

3 bit - 1 corner: $\mathbf{X} \in \{0,1\}^3$ — class 1 if $\sum_{i=1...3} x_i = 3$, else class 0

3 bit - 2 corners: $\mathbf{X} \in \{0,1\}^3$ — class 1 if $\sum_{i=1...3} x_i = 0$ or 3, else class 0

5 bit - 1 corner: $\mathbf{X} \in \{0,1\}^5$ — class 1 if $\sum_{i=1...5} x_i \leq 1$, else class 0

5 bit - 2 corners: $\mathbf{X} \in \{0,1\}^5$ — class 1 if $\sum_{i=1...5} x_i \leq 1$ or $\sum_{i=1...5} x_i \geq 4$, else class 0

We first examine tasks where all instances in the instance space are presented to the algorithm during learning. Here, functions are especially chosen to capture the inherent characteristics of different distributions of vectors in the instance space. We chose both linearly separable and non-linearly separable functions and also compared conjunctive versus k-of-n threshold concepts, as the former tend to be closely aligned with the principle axes of the instance space whereas the latter are not.

The three adaptive algorithms were trained using 10,000 instance presentations (drawn randomly) at each node of the tree during construction. Both the Bayes and ID3 algorithms were given an exhaustive enumeration of the instance space of the function to be learned. The error toleration parameter, E, was set to 0% as there was no noise

in the training instances. We averaged the algorithms over 10 runs. The accuracy and standard deviation when tested on the entire instance space is reported in Table 1. The average size of the trees produced is also shown in parentheses. Note that the non-adaptive schemes always induce the same decision tree given the same training data.

Algorithm	3 bit-1 corner	3 bit-2 corners	5 bit-1 corner	5 bit-2 corners
Perceptron	100.0±0.0 (1.0)	100.0±0.0 (2.0)	100.0±0.0 (1.0)	96.3±11.3 (4.7)
LMS	96.3±5.7 (1.7)	92.5±11.5 (2.4)	96.9±3.4 (3.9)	97.2±3.3 (12.2)
Back-Prop	100.0±0.0 (1.0)	100.0±0.0 (2.0)	100.0±0.0 (1.0)	100.0±0.0 (2.0)
N. Bayes	100.0±0.0 (1.0)	75.0±0.0 (2.0)	84.4±0.0 (3.0)	65.6±0.0 (3.0)
ID3	100.0±0.0 (3.0)	100.0±0.0 (5.0)	100.0±0.0 (14.0)	100.0±0.0 (23.0)

Table 1. Results on the function approximation tasks.

The results of these experiments show that not only do the adaptive learning methods equal or surpass the accuracy of classical methods for inducing decision trees, but these trees are also much smaller in size. The trees generated by Back-Propagation are not only perfectly accurate, but they reflect the minimum number of hyperplanes required to separate the instance space. Since these trees can now be tranformed into neworks, we see that it is possible to simultaneously learn both the topology of a network needed to learn a function and to learn the function itself. Moreover, we can use the TLU tree algorithm as an initial step in neural network design to give us an idea as to how large a network should be to learn a given function. We can first generate the appropriate TLU tree, transform it into a network with "informed" initial weights and then train the network using any network training method we wish.

4.3 The Monks Problems

For further learning experiments, we use a standard test-bed for learning tasks known as the Monk's Problems [TH, 1991]. This set of three learning tasks has been well studied on a number of different machine learning algorithms and includes standard training and test sets to help make fair comparisons between learning methods. For our learning algorithms we encoded these problems into a 17 dimensional boolean space using a local encoding scheme. As above, we used 10,000 samples drawn randomly with replacement from the training set to train the nodes of our TLU trees for the adaptive learning methods. The statistical learning methods had the entire training set available from which to compute frequency statistics. Since the first two Monks Problems are noise free and the third contains noise, we ran the tree induction algorithm using an error toleration parameter, E, of 0% for problems 1 and 2 and 10% for problem 3. Each experiment involved running each algorithm 10 times. Table 2 shows the accuracy and standard deviation of the induced TLU trees being tested on the entire instance space (much of which has not been seen before) as a test set. The averegage tree size is shown in parentheses.

Algorithm	Monk 1 (E = 0%)	Monk 2 (E = 0%)	Monk 3 (E = 10%)
Perceptron	83.9±6.3 (11.0)	75.5±5.3 (23.9)	95.9±1.6 (1.7)
LMS	92.2±3.5 (13.3)	76.9±6.5 (42.2)	94.2±2.5 (5.0)
Back-Prop	100.0±0.0 (3.0)	94.0±3.7 (7.4)	94.5±1.0 (1.1)
Naive Bayes	70.8±0.0 (5.0)	67.1±0.0 (2.0)	97.2±0.0 (1.0)
ID3	92.6±0.0 (20.0)	86.6±0.0 (45.0)	97.2±0.0 (2.0)

Table 2. Results on the Monks Problems learning tasks.

In considering the first two Monks Problems, we find that the TLU trees produced using Back-propagation seem to clearly outperform the non-adaptive methods when considering a combination of accuracy and tree size. On Monk 3 the results are a bit more inconclusive as all the learning methods seem to fare comparably in size and accuracy (possibly due to the error toleration parameter and noise in the training set). Interestingly, the trees induced using Perceptron and LMS are not nearly as small as expected, and their accuracy also seems to suffer as a result of not being able to learn a parsimonious multivariate shattering of the instance space. We conjecture that this problem results from these algorithms inability to converge properly when presented with non-linearly separable data, further complicated by the high dimensionality of the instance space, as we did not see this problem arise as severely when learning simple boolean functions. Nevertheless, the adaptive methods still fare well, led by Back-propagation which produces small and accurate (often the best) trees, especially in Monk 2 where the Back-propagation TLU trees are most accurate and yet 5 times smaller than trees induced with ID3.

5 Conclusions

We have shown the TLU tree algorithm to be a viable learning method capable of inducing small, yet accurate, decision trees by allowing each node to test more than one attribute. Not only can the TLU tree algorithm be used as a stand-alone learning algorithm, but the trees produced by it can be transformed into neural networks to determine a rough approximation of the topology necessary to properly learn a given data set. Moreover, by inducing a TLU tree, we can produce an initial set of weights for a network which we can further train incrementally as new training data becomes available.

Acknowledgments We are indebted to Nils Nilsson for providing the initial guidance and support to pursue this line of research. Discussions with George John and Pat Langley have also been useful. The author is supported by a Fred Gellert Foundation ARCS fellowship.

References

[BR, 1990] Brent, R.P. 1990. Fast training algorithms for multi-layer neural nets. Numerical Analysis Project Manuscript NA-90-03, Computer Sci. Dept, Stanford.

[BU, 1992] Brodley, C.E., and Utgoff, P.E. 1992. Multivariate Versus Univariate Decision Trees. COINS Technical Report 92-8, Computer Science Dept., UMass.

[BU, 1994] Brodley, C.E., and Utgoff, P.E. 1994. Multivariate Decision Trees. To appear in *Machine Learning*.

[GA, 1986] Gallant, S.I. 1986. Optimal Linear Discriminants. In *Eighth International Conference on Pattern Recognition*, 849-852. New York: IEEE.

[NI, 1965] Nilsson, N.J. 1965. *Learning machines*. New York: McGraw-Hill.

[QU, 1986] Quinlan, J.R. 1986. Induction of decision trees. *Machine Learning 1*:81-106.

[RHW, 1986] Rumelhart, D.E.; Hinton, G. E.; and Williams, R.J. 1986. Learning internal representations by error propagation. *Parallel Distributed Processing, Vol. 1*, eds. D. E. Rumelhart and J. L. McClelland, 318-62. Cambridge, MA: MIT Press.

[SA, 1993] Sahami, M. 1993. Learning Non-Linearly Separable Boolean Functions With Linear Threshold Unit Trees and Madaline-Style Networks. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 335-41. Menlo Park, CA: AAAI Press.

[TH, 1991] Thrun, S.B., and 23 co-authors. 1991. The monk's problems: a performance comparison of different learning algorithms. TR CMU-CS-91-197, Carnegie Mellon.

[WW, 1988] Widrow, B., and Winter, R.G. 1988. Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition. *IEEE Computer, March*:25-39.