Perceptron Trees: A Case Study in Hybrid Concept Representations

Paul E. Utgoff
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

The paper presents a case study in examining the bias of two particular formalisms: decision trees and linear threshold units. The immediate result is a new hybrid representation, called a *perceptron tree*, and an associated learning algorithm called the *perceptron tree error correction procedure*. The longer term result is a model for exploring issues related to understanding representational bias and constructing other useful hybrid representations.

1 Introduction

A core problem in machine learning is how to learn from examples. One would like to observe positive and negative instances of a concept, and be able to identify a generalization that is both correct for the observed instances and a good predictor for the classification of unobserved instances. Several algorithms have been devised, including Candidate Elimination [Mitchell, 1978], ID3 [Quinlan, 1983], AQ [Michalski and Chilausky, 1980], ID4 [Schlimmer and Fisher, 1986], and ID5 [Utgoff, 1988].

A fundamental issue in concept learning is the problem of built in biases that cause some generalizations to be preferred to others, even among those generalizations that are consistent with all the observed training instances [Utgoff, 1986]. This paper is concerned with biases that are inherent in a given concept formalism. Here, formalism is seen as one aspect of representation. Examples of formalisms include predicate calculus, formal grammars, set-theoretic notation, and other algebras. A second aspect of representation is the set of particular predefined terms and concepts that provide the basic building blocks for constructing concept descriptions within the formalism.

An example of bias that is inherent in a formalism is evident in the decision tree formalism. It is biased toward concepts that are expressed as boolean combinations of the instance features. If the concept to be learned is based on something other than a boolean combination, then the decision tree formalism will be a poor choice, resulting in a large tree that generalizes poorly for the unobserved instances. Consider the "numerically greater than" relation. A decision tree formalism would be a poor choice, because each ordered pair (x, y) would need to appear in the tree. The result would be rote learning.

It is beyond the scope of this paper, and beyond our present knowledge, to make any catalogue of formalisms and their inherent biases, or to draw any large conclusions about such biases. Instead, this paper presents a case study in examining the bias of two particular formalisms: decision trees and linear threshold units. The immediate result is a new hybrid representation, and an associated learning algorithm. The longer term result is a model for exploring issues related to understanding representational bias and constructing other useful hybrid representations.

2 Motivation

The thesis of the work is that individual concept formalisms have inherent biases. This implies that no single formalism is the best choice for all concept learning problems. It would increase the autonomy and effectiveness of a learning program if it were able to make its own choices regarding selection of formalism. Such choices should occur at every level of the learning, including terms or subconcepts, not just at the top level. The result of mixing formalisms and statements within those formalisms is a *hybrid representation*. By selecting an appropriate formalism for each subconcept, the learning program draws on the special strengths of that formalism. *Strength* is used loosely to refer to the ease with which particular concepts can be described within the formalism. To the extent that the strength of each individual representation complements the weaknesses of the others, the hybrid representation is enriched.

The present work arose from the need for a learning program to be able to handle a stream of training instances flowing at a rate of up to several thousand instances per minute [Utgoff and Heitman, 1988]. In terms of handling a large volume of instances, decisions tree methods and connectionist learning methods are natural choices.

The desire to find concepts that are consistent with *all* the training instances, given that the training instances are labeled consistently[1], favored decision trees, leading to examination of ID3 [Quinlan, 1983], ID4 [Schlimmer and Fisher, 1986], and ID5 [Utgoff, 1988]. Unfortunately, ID3 is not incremental, ID4 does not always find a consistent concept description, and ID5 saves the training instances, making it space-inefficient for a large volume of instances. More important, the bias of the decision tree formalism was inappropriate for many kinds of concepts that were to be learned.

Connectionist methods provide the needed efficiency in handling training instances, but there is no existing theory regarding choice of network architecture. This is of critical importance, since choice of network is closely analogous to the problem of selecting a representation. For example, a

[1] i.e. any given training instance is always classified the same way on every presentation.

large network can represent more concepts than a small network. The choice of network architecture directly affects the expense of updating weights, granularity of representation, and quality of generalization.

The following requirements and assumptions emerged:

1. The learning algorithm must be able to handle a large volume of training instances efficiently and incrementally.
2. The algorithm must be able to select an appropriate formalism at any level.
3. The algorithm must find a consistent concept description in finite time without human intervention.
4. The training instances are assumed to be labeled consistently.
5. The resulting concept description must be efficient for classifying unobserved instances.

Now consider the characteristics of learning with the decision tree formalism and learning with the linear threshold unit formalism.

3 Decision Trees and Linear Threshold Units

A *decision tree*, especially as described by Quinlan, is a node that contains an answer (typically '+' or '−' to indicate the classification) or an attribute test with, for each value that the attribute can take on, a branch to a decision tree. Each branch represents a disjunction. Each distinct path through the tree, from the root to an answer node, represents a conjunction. A decision tree can be viewed as a factored boolean expression. For classification purposes, a decision tree is traversed, starting at the root, according to the decision nodes in the tree and the corresponding values in the instance, until an answer node is reached. There is a large literature on methods for constructing decision trees [Moret, 1982]. Throughout this paper, the information-theoretic approach is assumed [Lewis, 1962; Quinlan, 1983], in which the tree building process selects the attribute test that removes the greatest amount of ambiguity, leaving the least amount of expected decision making to be done. This kind of tree building procedure is quasi-optimal. The structure of a decision tree has the effect of partitioning the instance space at each decision node, due to the manner in which the tree is traversed for classification purposes.

A *linear threshold unit*, herein abbreviated LTU, is a device that compares a weighted sum of instance features to a fixed threshold value [Minsky and Papert, 1969]. The model assumes that presence or absence of a feature in an instance is represented numerically. An instance is represented by a vector $\mathbf{I}$ that encodes the presence or absence of each feature. The LTU maintains one weight for each feature and one weight for the constant term (viewing the weights as coefficients of a linear polynomial), making a vector $\mathbf{W}$ of such weights. The constant term, denoted θ throughout this paper, is treated as a feature that is present in every training instance. If $\mathbf{W} \cdot \mathbf{I}$ is greater than or equal to 0 then the instance $\mathbf{I}$ is classified as positive by the LTU. Otherwise, the LTU classifies $\mathbf{I}$ as negative. Geometrically, $\mathbf{W}$ defines a hyperplane. The inner product of $\mathbf{W}$ and $\mathbf{I}$ indicates which side of the hyperplane $\mathbf{I}$ is on. As guaranteed by the Perceptron Convergence Theorem [Minsky and Papert, 1969], a $\mathbf{W}$ that separates the positive and negative instances via a hyperplane can be found in a finite number of steps if such a $\mathbf{W}$ exists. This means that a LTU will find a consistent concept description if and only if the target concept is describable by a hyperplane.

	Decision Tree	LTU
Complete Representation	Yes	No
Guaranteed Convergence	Yes	No
Efficient Update	No	Yes
Efficient Classifier	Yes (very)	Yes
Boolean Combination Bias	Yes	No
Hyperplane Bias	No	Yes

Table 1: Characteristics of the Two Methods

Now consider the characteristics of the two formalisms and their associated learning algorithms as listed in table 1. The item "Complete Representation" refers to whether every concept over the instance space is representable. "Guaranteed Convergence" indicates whether the associated learning algorithm is guaranteed to find a concept description that is consistent with all the observed training instances. The item "Efficient Update" refers to the expense of handling a training instance that has been presented to the learning algorithm.

In qualitative terms, one would like a representation and associated learning algorithm that possesses all the favorable and none of the unfavorable characteristics. Note that neither decision trees nor LTUs alone possess all the favorable characteristics.

4 Perceptron Trees

This section reports a case study in constructing a hybrid representation and associated learned algorithm. It is motivated by the requirements listed above in section 2 and by the observation in table 1 that decision trees and linear threshold units complement each other well.

4.1 Perceptron Tree Representation

Define a *perceptron tree* to be either a linear threshold unit, or an attribute test with, for each value the attribute can take on, a branch to a perceptron tree. The term "perceptron tree" was chosen because the linear threshold unit is the basic unit of Rosenblatt's perceptron. A perceptron tree is much like a decision tree, except that every leaf node is a LTU. As explained below, this is not simply a case of trading in answer nodes for LTUs. Given the ability of a LTU to represent concepts, a LTU can serve in place of a decision tree or subtree. The number of decision nodes in a perceptron tree need never exceed the number of nodes in a plain decision tree, and will typically be less.

For the work reported here, the *symmetric* model of instance representation is assumed [Hampson and Volper, 1986]. Each feature is represented by 1 if present in the instance and -1 otherwise. A *feature* is a specific value of a specific attribute. For example, if the color of the instance is red, then the attribute is "color", the value is "red", and the feature is "color is red".

It is important to note that the perceptron tree formalism is complete.

Theorem 1 *The perceptron tree formalism is complete in the sense that for every possible subset of the instance space, there is a perceptron tree that can describe exactly that subset.*

Proof: The decision tree formalism is complete. Because a perceptron tree could be elaborated to a plain decision tree down to the point that each instance is described by a single attribute, it is sufficient to show that a LTU can discriminate instances described by a single feature. This is trivially so because for each attribute value i observed in some positive instance, weight $w_i = 0$ will cause that instance to be classified positive. Similarly, for each attribute value i observed in some negative instance, weight $w_i = -1$ will cause that instance to be classified negative. Under the assumption that a training instance is never labeled positive on one occasion and negative on another, it will always be the case that an instance with a given value of the attribute can be uniquely classified. □

A perceptron tree is a hybrid representation. It is a disjunction of hyperplanes, each selected by a unique conjunction of features. A perceptron tree may need to be a decision tree down to the point that each leaf LTU is discriminating instances based on a single attribute. This would be equivalent to a complete decision tree. However, a perceptron tree offers the ability to describe a space of instances with a LTU. A perceptron tree can be smaller than a plain decision tree in terms of number of decision nodes. This is an advantage because the need to partition the instance space on an attribute test may be significantly reduced. It is potentially a disadvantage if obtaining the value of each attribute is considered expensive, because a LTU requires obtaining the value of every attribute not tested above in the tree. In terms of computation, obtaining all the values can be done in parallel. However, depending on the application, it could be expensive or unwarranted to perform all the tests.

4.2 Perceptron Tree Error Correction Procedure

The error correction procedure incrementally updates a perceptron tree, which is a global data structure. The initial perceptron tree consists of a single empty node. An *empty node* is a node that contains no information and has not yet been initialized as either as decision node or a LTU node. A *decision node* is a node that contains an attribute test and, for each value of the test attribute that has been observed previously, a branch to a perceptron tree. A *LTU node* is a node that contains a linear threshold unit. The notation $dim(\mathbf{W})$ indicates the number of components (features) in vector $\mathbf{W}$. The procedure that updates a perceptron tree in response to a training instance, called the *perceptron tree error correction procedure*, is:

1. While at a decision node, traverse the indicated value branch.
2. If at a decision node, then there was no value branch corresponding to the value in the instance. Add a new branch with a new empty node at its leaf and traverse the branch to the leaf.
3. If at an empty node, then make it a LTU node and initialize the LTU at the node. The LTU is initialized by setting all weights in the vector $\mathbf{W}$ of the LTU to 0. Any other bookkeeping variables for the LTU are also initialized.
4. Compute the relationship of instance $\mathbf{I}$ to the hyperplane defined by $\mathbf{W}$ by $y \leftarrow \mathbf{W} \cdot \mathbf{I}$.
5. If $y \geq 0$ and the training instance is negative, then adjust $\mathbf{W}$ so that $\mathbf{I}$ would have been correctly classified as negative. This is computed by $\mathbf{W} \leftarrow \mathbf{W} - \mathbf{I} \cdot (\lfloor \frac{y}{dim(\mathbf{W})} \rfloor + 1)$. Go to step 7.
6. If $y < 0$ and the training instance is positive, then adjust $\mathbf{W}$ so that $\mathbf{I}$ would have been correctly classified as positive. This is computed by $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{I} \cdot (\lfloor \frac{-y-1}{dim(\mathbf{W})} \rfloor + 1)$.
7. If the space of instances at this node should be partitioned into subspaces (explained below), then discard the LTU at this node and replace it with an attribute test. This makes the node a decision node with no branches. (There is no immediate need to provide branches below the node because they will be grown as necessary with subsequent training.)

There are four points to note. First, the procedure indicated for adjusting $\mathbf{W}$ in steps 5 and 6 above is the *absolute error correction procedure* described in Nilsson [Nilsson, 1965]. Second, $\mathbf{W}$ is integer-valued. Third, a perceptron tree only grows, it never shrinks. Finally, the $\mathbf{W}$ at each LTU corresponds to the features that were not determined by decision nodes. For example, if "color" is a test attribute above a given LTU, then no feature corresponding to "color" is part of the $\mathbf{W}$ of that LTU. This is because the attribute "color" and its value are fixed as a result of taking that path through the decision nodes of the perceptron tree.

There are two issues in step 7 above. First is the problem of detecting when the space of instances should be partitioned via an attribute test. The second is the problem of picking the attribute for the decision node of the perceptron tree. A specific method for deciding when to partition, and a specific method for picking an attribute are given below. Together, they illustrate one way of instantiating step 7 of the procedure. The sole requirement is that the space of instances at a node be split if that space is not linearly separable.

4.2.1 When to split

If the space of instances at a node is not linearly separable, then it is necessary that the space be split (partitioned) into subspaces. A space of instances is *linearly separable* if there exists a hyperplane that discriminates the positive and negative training instances. The problem is to detect that the space of instances is not linearly separable. The Perceptron Cycling Theorem [Minsky and Papert, 1969] states that the perceptron learning algorithm visits a finite number of weight vectors $\mathbf{W}$, assuming integer valued weights, regardless of separability. A corollary [Gallant, 1986] is that the perceptron learning algorithm will leave and revisit at least one weight vector if and only if the space of instances is not linearly separable. Thus, to prove nonlinear separability, it is sufficient to prove that the current weight vector $\mathbf{W}$ has been visited before. A sufficient test for separability is:

Corollary 1 *If the number of vectors visited (so far) exceeds the number of distinct vectors that could have been*

visited (so far), then the space of instances is not linearly separable.

To be able to compute an upper bound on the number of distinct vectors that could have been visited so far, the minimum and maximum value that each weight w_i has ever taken on are maintained within the LTU. The notations $w_{i,min}$ and $w_{i,max}$ indicate, respectively, the minimum and maximum value w_i has ever taken on. An upper bound on the number of distinct vectors that could have been visited is:

$$\prod_{i=1}^{dim(\mathbf{W})} (w_{i,max} - w_{i,min} + 1) \qquad (1)$$

This leads immediately to:

Corollary 2 *Nonlinear separability can be detected in a finite number of steps, without saving previous weight vectors.*

This follows immediately because the above upper bound on the number of distinct weight vectors that could have been visited is finite. Thus, by corollary 1 and the above computable upper bound (1) on the number of distinct vectors that could have been visited so far, a procedure exists for detecting nonlinear separability: if the number of vectors visited (so far) exceeds upper bound (1), then the space of instances is not linearly separable.

The test for nonlinear separability is correct, but conservative because the upper bound is not tight. Cycling can occur long before the test detects it. A test is needed that both detects cycling when it first occurs and does not require saving the training instances. Gallant's "Pocket Algorithm" [Gallant, 1986] addresses the problem indirectly by detecting when the classification performance of a best weight vector for a linear threshold unit appears to have reached an asymptote. Although such a test does not prove nonlinear separability, it may provide a good heuristic. Ho and Kashyap [Ho and Kashyap, 1965] constructed a procedure for detecting an inconsistency in a set of linear inequalities, but it requires saving the training instances.

For the current work, a more aggressive test is used for deciding when to split. Due to the completeness of the perceptron tree representation, splitting more often than is strictly necessary is not harmful, in the sense that the ability to find a consistent concept description is not lost. It means that it is possible that a decision node will have split the space even though a LTU would have been sufficient. Instead of detecting only nonlinear separability, the test detects when the LTU is not making significant progress toward arriving at a consistent concept description.

The test is based on the number of vector adjustments of $\mathbf{W}$ that have occurred since some $w_{i,min}$ or some $w_{i,max}$ has been adjusted. If $\mathbf{W}$ continues to be adjusted in response to misclassified training instances, yet the minimum and maximum values of the w_i come to be adjusted rarely or seemingly not at all, then there is reason to believe that there is lack of progress in moving toward a solution vector. At issue is how many weight adjustments without changing a $w_{i,max}$ or a $w_{i,min}$ constitute lack of progress. Let C be the number of consecutive vector adjustments to $\mathbf{W}$ since some $w_{i,min}$ or some $w_{i,max}$ has been adjusted. The test is: if $C > dim(\mathbf{W})$ then split the space of instances.

4.2.2 Where to split

The problem of picking an attribute test for a decision node has received much attention in the fields of pattern recognition and statistics [Fu, 1968; Moret, 1982]. As mentioned above in section 3, the approach taken here is to employ an information-theoretic criterion that measures the amount of ambiguity in a space of instances. The attribute that removes the greatest amount of ambiguity, by partitioning the space into the least ambiguous subsets, is chosen as the attribute test for the decision node. See Quinlan [Quinlan, 1983] for a specific algorithm. See section 3.3.1 of Moret [Moret, 1982] for a general discussion and for references to theoretical work.

The information-theoretic splitting criterion currently in use requires the number of positive and number of negative instances observed for each of the w_i. These counts are maintained in each LTU, and are updated for every observed training instance, whether or not the weights in $\mathbf{W}$ are adjusted.

4.2.3 Convergence to a Consistent Concept Description

Given that there exists a perceptron tree representation of a concept description that is consistent with all the training instances, one needs to consider whether such a description will be found.

Theorem 2 *If the training instances are labeled consistently, then the perceptron learning algorithm, using the perceptron tree error correction procedure, will find a consistent concept description in a finite number of steps.*

Proof: Either the LTU finds a solution vector in a finite number of steps, as per the Perceptron Convergence Theorem, or the space of instances is detected to be not linearly separable in a finite number of steps (corollary 2). If the space of instances is not linearly separable, then it is split with an attribute test. Since the algorithm is applied recursively at each node, it is only necessary to show that a linearly separable space is finally reached at each LTU node. This is guaranteed by the completeness of the representation (theorem 1) and the consistent labeling assumption. □

4.2.4 Learning Behavior

Because a perceptron tree has a LTU at each leaf node, much of the learning behavior is characteristic of a LTU. As per the perceptron learning algorithm, one must repeatedly present the training instances because the LTU is *not* guaranteed to remain consistent with the previously observed training instances. A perceptron tree has the additional characteristic that replacing a leaf LTU with a decision node (attribute test) causes that LTU to be discarded. New LTUs must be trained at each new leaf node below the new decision node. This is most noticable when the initial root LTU is replaced by an attribute test. The effect becomes less noticable at subsequent splits because the rest of the perceptron tree remains intact.

4.3 An Illustration

To illustrate various characteristics of learning with the perceptron tree error correction procedure, a simple problem was formulated. The problem is to learn the concept

$$(a \vee b) \oplus (c \wedge d)$$

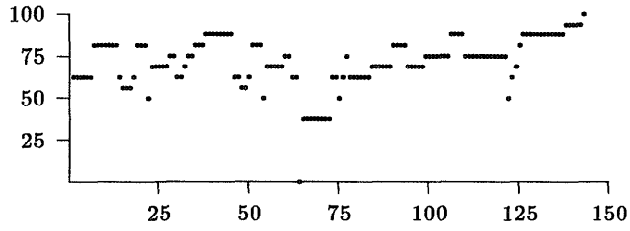

Figure 1: Percent correct (y axis) vs instances.

where a, b, c, and d are boolean, and $\oplus$ indicates exclusive-or. There are only 16 possible instances. An instance is an example of the concept if and only if $(a \vee b) \oplus (c \wedge d)$ is true for the given values of a, b, c, and d in the instance. This problem was chosen because the concept cannot be learned by a single LTU and because the subconcepts involve testing whether some x of n variables are true, a kind of problem which is well suited to a LTU.

The standard perceptron learning algorithm repeatedly draws a training instance at random from the set of training instances, and presents it to the error correction procedure in use. A variant of the algorithm was employed here, in which the training instances were drawn in order from the entire space of 16, one after the other. The list of training instances is considered to be circular.

The training procedure was: while the perceptron tree fails to classify all 16 instances correctly, apply the perceptron tree error correction procedure to the next training instance. Figure 1 shows the percentage of the 16 training instances classified correctly after training on the next training instance. The first split occurred while training on the 64th instance. Classification performance temporarily dipped to 0% when the perceptron tree consisted of a decision node with no branches. As the branches were grown on subsequent training, performance was generally better than before the split. The second split came while training on the 122nd instance. Classification performance temporarily dipped to 50% because one of the leaf nodes was a decision node with no branches. As the branches below that node were grown during subsequent training, performance climbed to 100% after the 143rd instance.

Figure 2 shows the final perceptron tree. It contains 2 decision nodes and 3 LTU nodes. Each LTU is depicted as a simple matrix in which the row is indexed by the value of a variable and the column is indexed by the name of the variable. To illustrate the LTU notation, how a symmetric LTU operates, and how a perceptron tree is used to classify an instance, consider how the instance ($a = F$, $b = T$, $c = T$, $d = F$) is classified. Because a is the test attribute at the root, and $a = F$ in the instance, the F branch is taken. Because b is the test attribute at the subtree, and $b = T$ in the instance, the T branch is taken. Now, at the LTU node, the instance is encoded as 1 for each feature present and -1 for each feature absent. Thus $\mathbf{W} \cdot \mathbf{I}$, i.e.

	c	d	θ
F	1	0	
T	−1	−1	1

$\cdot$

	c	d	θ
F	−1	1	
T	1	−1	1

$= 0$

so the instance is classified as positive.

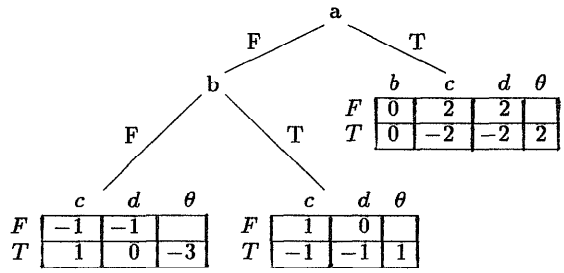

Figure 2: Perceptron tree

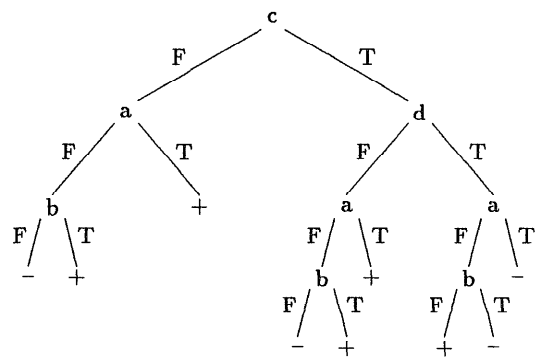

Figure 3: Decision tree

Figure 3 shows the plain decision tree that would be built by Quinlan's ID3. Note that it has 8 decision nodes and 9 answer nodes.

5 Related Work

An alternative method for combining decision trees and LTUs has been proposed in [Breiman et al, 1984]. Their approach is to place a LTU at each decision node. If, for a set of weights $\mathbf{W}$, $\mathbf{W} \cdot \mathbf{I}$ is greater than or equal to 0, then branch one way, else branch the other. Leaf nodes are answer nodes, indicating either that the instance is a positive instance or that it is a negative. This approach if different from perceptron trees, in which each decision node contains an attribute test, and each leaf node contains a LTU.

Schlimmer [Schlimmer, 1987] has constructed a hybrid representation and associated learning algorithm embodied in his STAGGER program. The program maintains a pair of weights for each boolean term in his concept description. One corresponds to logical sufficiency, the other to logical necessity. By adjusting the weights and by adding or removing boolean terms, the program searches for a consistent concept description. A recent addition to STAGGER is the ability to group values of real-valued attributes into dynamically formed intervals, which constitute new boolean terms that can become part of the concept description.

6 Conclusions

The perceptron tree representation and the perceptron tree error correction procedure offer a new mechanism for concept learning. The immediate result can be seen either as a method for perceptron learning even when the space of instances is not linearly separable, or as a method for incremental construction of a tree structure that is very much like a decision tree. The algorithm is incremental, does not save training instances, and is guaranteed to find a consistent concept description for all problems in which the instances are labeled consistently.

An analysis of learning rate is still lacking. For every call to the perceptron tree error correction procedure, some number (possibly 0) of decision nodes will be traversed until a LTU node is reached, at which point the LTU is updated using the symmetric feature representation and the absolute error correction procedure. For some training events, a LTU will be discarded and replaced by an attribute test. This kind of activity is low compared to the total time spent in updating some LTU, so such activity can be discounted. Thus, it seems that much of the theoretical analysis regarding rate of convergence for learning with a single LTU, for linearly separable sets, would apply, but this has not been established. See Hampson and Volper [Hampson and Volper, 1986] for a recent analysis of learning rate using LTUs.

The work has been motivated by the specific need for an efficient incremental learning algorithm, and by the observation that the inherent biases in the formalisms of two efficient learning algorithms are highly complementary. The ease of incrementally training a linear threshold unit complements the difficulty of incrementally building a decision tree. The ability to represent any concept in the decision tree formalism complements the inability to represent not linearly separable concepts in the hyperplane formalism. The combination of complementary formalisms into a hybrid makes it possible to draw on the particular strengths of each of the individual formalisms. The case study reported here demonstrates that a perceptron tree representation retains the advantages of both the decision tree representation and the hyperplane representation, while shedding the major disadvantages.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IRI-8619107 and by a General Electric Faculty Fellowship. Helpful comments have been provided by Andy Barto, Sharad Saxena, Peter Heitman, Margie Connell, Jamie Callan, Kishore Swaminathan, Victor Coleman, and Richard Yee.

References

[Breiman et al, 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984) *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

[Fu, 1968] Fu, K. S. (1968) *Sequential methods in pattern recognition and machine learning.* Academic Press.

[Gallant, 1986] Gallant, S. I. (1986) Optimal linear discriminants. In *Proceedings of the International Conference on Pattern Recognition* (pp. 849-852). IEEE Computer Society Press.

[Hampson and Volper, 1986] Hampson, S. E. and Volper, D. J. (1986) Linear function neurons: Structure and training. In *Biological Cybernetics, 53,* 203-217. Springer-Verlag.

[Ho and Kashyap, 1965] Ho, Y. C. and Kashyap, R. L. (1965) An algorithm for linear inequalities and its applications. *IEEE Transactions on Electronic Computers, EC-14(5),* 683-688.

[Lewis, 1962] Lewis, P. M. (1962) The characteristic selection problem in recognition systems. *IRE Transactions on Information Theory, IT-8(2),* 171-178.

[Michalski and Chilausky, 1980] Michalski, R. S. and Chilausky, R. L. (1980) Learning by being told and learning from examples. *Policy Analysis and Information Systems, 4(2).*

[Minsky and Papert, 1969] Minsky, M. and Papert. S. (1969) *Perceptrons: An introduction to computational geometry.* MIT Press.

[Mitchell, 1978] Mitchell, T. M. (1978) *Version spaces: an approach to concept learning.* Ph.D. dissertation, Stanford University. (also Stanford CS report STAN-CS-78-711, HPP-79-2).

[Moret, 1982] Moret, B. M. E. (1982) Decision trees and diagrams. *Computing Surveys, 14,* 593-623.

[Nilsson, 1965] Nilsson, N. J. (1965) *Learning machines.* McGraw-Hill.

[Quinlan, 1983] Quinlan, J. R. (1983) Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell, & Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 463-482). Morgan Kaufmann.

[Schlimmer and Fisher, 1986] Schlimmer, J. C. and Fisher, D. (1986) A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 496-501). Morgan Kaufman.

[Schlimmer, 1987] Schlimmer, J. C. (1987) Learning and representation change. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 511-515). Morgan Kaufman.

[Utgoff, 1986] Utgoff, P. E. (1986) Shift of bias for inductive concept learning. In Michalski, Carbonell, & Mitchell (Eds.), *Machine learning: An artificial intelligence approach, II* (pp. 107-148). Morgan Kaufmann.

[Utgoff and Heitman, 1988] (1988) Utgoff, P. E. and Heitman, P. S. Learning and generalizing move selection preferences. In *Proceedings of the AAAI Symposium on Computer Game Playing* (pp. 36-40).

[Utgoff, 1988] (1988) Utgoff, P. E. ID5: An incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufman.