Appears in *Proceedings of the International Joint Conference on Neural Networks (IJCNN-92)*, Baltimore, MD, vol. 2, pp. II-392 - II-397, June, 1992

Growing Layers of Perceptrons: Introducing the Extentron Algorithm

Paul T. Baffes and John M. Zelle

Department of Computer Sciences,
University of Texas at Austin,
Austin, Texas 78712
baffes@cs.utexas.edu, zelle@cs.utexas.edu

Abstract

The ideas presented here are based on two observations of perceptrons: (1) when the perceptron learning algorithm cycles among hyperplanes, the hyperplanes may be compared to select one that gives a best *split* of the examples, and (2) it is always possible for the perceptron to build a hyperplane that separates *at least one* example from all the rest. We describe the Extentron which grows multi-layer networks capable of distinguishing non-linearly-separable data using the simple perceptron rule for linear threshold units. The resulting algorithm is simple, very fast, scales well to large problems, retains the convergence properties of the perceptron, and can be completely specified using only two parameters. Results are presented comparing the Extentron to other neural network paradigms and to symbolic learning systems.

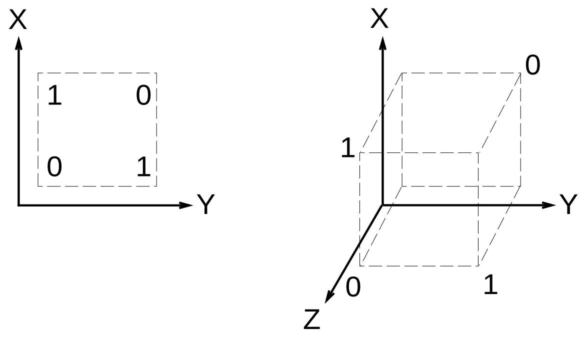

Figure 1: **Explicit representation of a hyperplane.** Example of a hyperplane for the XOR problem. On the left is a drawing of the original four examples given to the perceptron. Each is labeled with a "1" (positive) or "0" (negative). On the right is a mapping of the same examples after extension.

1 Introduction

It is well known that the simple perceptron algorithm (Rosenblatt, 1958) is unable to represent classifications which are not linearly separable (Minsky and Papert, 1988). However, the perceptron does exhibit two very useful properties. First, since there are no hidden layers, designing the topology of the network is not an issue for the user. One simply uses the size of the input-output pairs to determine the form of the network. Second, it has been shown that the perceptron learning algorithm will either converge (if the data is linearly separable) or will cycle among a series of hyperplanes which do not fully separate the input examples. Additionally, when the perceptron learning algorithm cycles it is always possible for the perceptron to build a hyperplane that separates *at least one* example from all the rest. By taking advantage of these observations, we have been able to develop an algorithm which is simple, very fast, scales well to large problems, retains the convergence properties of the perceptron, and can be completely specified using only two parameters. But most important of all, this new algorithm can classify data which is not linearly separable, thus avoiding the critical limitation of the perceptron.

At the foundation of our algorithm is the concept of the hyperplane. A hyperplane found by a perceptron is implicit; it is stored in the weights connecting input nodes to output nodes. However, the values of the output nodes may be used as an *explicit* form of the hyperplane. Figure 1 shows an example of how this can be done for the XOR problem. Initially, the examples for XOR are not linearly separable. One possible hyperplane which may be found by a perceptron for this problem is one in which both of the positive examples are labeled correctly, but only one negative example (upper right in the figure) is labeled correctly. Using this hyperplane, the output of the perceptron for both positive examples and the lower left negative example is "1," whereas the output for the upper right example is "0."

The right half of figure 1 shows what happens to the input space when the examples are *extended* using the corresponding output of the perceptron. Here, the input for each example is now three bits long, with the z-axis representing the added bit. The examples which were on the same side of the hyperplane are separated *orthogonally* in the new input space from those examples on the other side of the hyperplane. The result, in this particular case, is a new set of examples which are linearly separable and can be learned by a simple perceptron.

In the following sections we describe the Extentron algorithm, which builds upon this idea of example extension. The goal of the Extentron is to grow multi-layer networks capable of distinguishing non-linearly-separable data using the simple perceptron rule for linear threshold units. Results are presented comparing the Extentron to other neural network paradigms and to symbolic learning systems.

2 Overview of Extentron

2.1 Basic Algorithm

The Extentron is an iterative algorithm. Initially, a perceptron is trained on the original data, which is assumed to be a list of examples in the form of input-output pairs. If the perceptron can successfully separate all the examples then no iteration need occur. Otherwise, the perceptron representing the best split of the examples is saved, and the examples are extended using the output of that perceptron as described above. The whole process is then repeated until all of the examples can be correctly classified.

More formally, let E be a set of examples and let E_p be the input-output pair for example p. Let I_p and T_p be the input and output (target) of the pair, respectively. Let H_E be the perceptron with the best hyperplane relative to the examples in E; i.e., the set of weights that allows the perceptron to correctly classify the most examples. Finally, let O_p be the output generated by H_E for example p. The basic Extentron algorithm is as follows:

1. Compute H_E using perceptron learning on the input examples E. If all examples are classified correctly, i.e. $O_p = T_p$ for all pairs E_p, then quit.

2. Otherwise, for each pair p in E, extend the input of the pair by appending the output O_p (generated by H_E) to the end of I_p. This forms a new set of examples E'.

3. Repeat with E' until all examples are correctly classified.

After training, the network returned by the Extentron is a series of layers, each of which is a perceptron that depends upon the preceding layer for its input. Propagating through an Extentron network is achieved by normal perceptron propagation, using the output of each layer to extend the input before propagating through the next layer. Figure 2 shows an Extentron net for the XOR problem.

2.2 Observations

Several important characteristics about the Extentron algorithm should be pointed

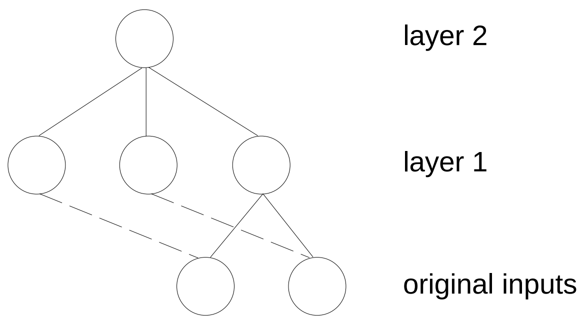

Figure 2: **Extentron network for the XOR problem.** This network consists of two layers, each of which is a perceptron. Solid lines represent perceptron weights. Dotted lines are "pass through" weights which are not learned. Here, the original input passes through to the second perceptron, extended by the output of the first perceptron.

out. First, note that the Extentron is guaranteed to converge on consistent training examples since each perceptron will generate a hyperplane that can separate at least one of the examples. Second, figure 1 implies that one extension of a set of examples will make them linearly separable, but in general this is not true. In the worst case, each perceptron could isolate only one example, causing the Extentron to grow a layer for each input-output pair. However, we have found empirically that an Extentron needs only a small number of layers (see section 5), unless the problem is pathologically difficult (e.g. the two-spiral problem).

Third, the structure of the network can be optimized (i.e., weights removed) for problems with more than one output bit. The savings comes from the fact that each output node effectively represents a separate perceptron, since its weights are updated without regard to the accuracy of the other output nodes. Figure 3 shows a network with two outputs, one of which (the rightmost) is completely learned by the first perceptron. The first extension of the examples (middle network) uses both outputs, causing the input space to go from two to four dimensions. But the second extension of the examples need not include the rightmost output node, since this information would be redundant. Consequently, the second extension of the examples causes the input space to go to five dimensions rather than six. Furthermore, once an output node is learned it can be omitted from the training phase. Thus, no weights (solid lines) are connected to the rightmost node after the first layer. Instead the correct value of this node is simply passed through (dotted lines) from the first layer.

Fourth, the Extentron only requires the user to set two parameters. One of these, *accuracy*, measures the degree to which the training examples must be accurately predicted. The other, *max-epochs*, is used as a heuristic for cycle detection.

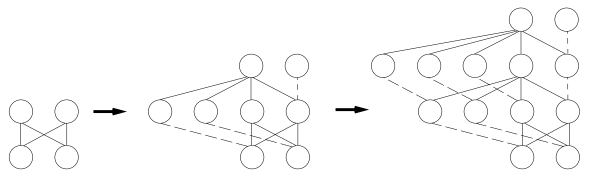

Figure 3: **Optimizing an Extentron network.** This diagram shows a three layer network for examples with two input bits and two output bits. The first layer completely learns the rightmost output bit, thus no weight connections are created to that output bit in succeeding layers and its value is passed through from the first layer.

Recall that the perceptron either converges or cycles. In general, it may take a very long time for a perceptron to cycle, and it is costly in both time and space to save all states of the weight vector and check for duplication after each epoch. The Extentron's solution to this problem is twofold. Initially, training epochs are performed until a unit's accuracy exceeds that of the corresponding unit of the previous layer. In the case of the first layer, a unit must do better than random guessing. This ensures that each layer is making progress towards correctly separating the examples. Once the initial hyperplane is found, the Extentron keeps track of the number of epochs for which there has been no further improvement in predictive accuracy. If this exceeds max-epochs, a cycle is assumed and a new layer is grown. Note that larger values of max-epochs will tend to create networks of fewer layers, but each layer will take longer to train.

Finally, the perceptron learning rule is very simple. This, combined with the weight optimization described above, make the Extentron's learning speed very fast compared to other neural network algorithms such as back propagation. For example, the XOR network of figure 2 above was produced in a fraction of a second using roughly 30 epochs. Detailed results are presented in section 4.1.

3 Previous Work

A hybrid system known as *perceptron trees* (Utgoff, 1988; Utgoff and Brodley, 1990) is the work most closely related to the ideas presented here. A perceptron tree is a decision tree, where each node of the tree is a linear threshold unit (perceptron). The inner nodes of the tree are used as decision nodes to split examples along different paths of the tree. Leaf nodes are perceptrons that can completely categorize some subset of the training examples. A perceptron tree begins as a single perceptron, only growing nodes when that perceptron fails to categorize all of its training examples. Each internal node of the perceptron tree effectively splits the training set, sending some of the examples to one child node and the rest to the other.

The chief difference between the Extentron and perceptron trees is that the Extentron is a strictly neural model, whereas a perceptron tree is a hybrid. Instead of splitting the examples into subsets, the examples are extended and *all* are reused to train the next perceptron. This allows the Extentron to avoid learning duplicate representations of the same feature. Specifically, the internal nodes of a perceptron tree are unable to use the information from other parts of the tree. Thus two subtrees of the decision tree may be forced to relearn the same hyperplane split. In short, by extending the inputs, each layer of an Extentron network is able to use any of the features learned to that point.

Back propagation (Rumelhart et al., 1986) can be compared to the Extentron using several criteria. Back propagation is a more general algorithm, due to the fact that real-valued outputs can be generated (as opposed to the binary outputs of the simple perceptron). Also, for pathologically non-linearly-separable domains such as the two-spiral problem, back propagation will tend to learn simpler networks since the Extentron will have to grow an excessive number of layers (hyperplanes) to successfully categorize the data. However, back propagation is not guaranteed to converge on its training data, and requires the user to adjust a large number of parameters by hand. The Extentron avoids both of these problems. Finally, due to the simpler learning rule, the Extentron is much faster than back propagation (see section 4.2), and yet retains similar generalization capabilities.

Finally, there are numerous symbolic classification algorithms which can be used to perform the same tasks as an Extentron. One such example is Quinlan's ID3 algorithm (Quinlan, 1986). As with perceptron trees, ID3 forms a decision tree to classify the input examples. However the internal nodes of ID3 use an information theoretic measure to determine a split in the data. Also, ID3's decision nodes are limited to using one feature of the input vector, as opposed to the multi-variate split available to both perceptron trees and the Extentron. Finally, ID3 decision trees suffer from the same potential duplication of effort as perceptron trees due to the lack of information sharing between internal nodes.

4 Experimental Method

4.1 Experimental Design

As an initial experiment, we tested the Extentron on some of the well known neural network problems. For the XOR problem, the Extentron completed training in 0.03 seconds running on a SPARC Station 1 under Sun Common Lisp (version 4.0). We also ran an Extentron on Lang and Witbrock's two-spiral problem (Lang and Witbrock, 1988). The Extentron managed to completely learn the examples, but only after growing 65 layers.

To test the Extentron against other classification algorithms, we duplicated some of the experiments run by Mooney et. al. (Mooney et al., 1990) and Shavlik et. al. (Shavlik et al., 1991). Two different data sets were used in these experiments, one for soybean diseases (Reinke, 1984) and the other for DNA promoter sequences (Hawley and McClure, 1983). These data sets involve, respectively: 50 and 57 features; 562 and 106 examples; 15 and 2 categories. After translation into a format usable by an Extentron the soybean data set had 79 inputs and the DNA data set had 228 inputs. The soybean data set is known not to be linearly separable (though subsets of this data are linearly separable), and the DNA data set has been shown to be difficult for symbolic systems to learn (Fisher and McKusick, 1989).

The experiments proceeded as follows. Each data set was divided into training and test sets. Training sets were further divided into subsets, so that the algorithms could be evaluated with varying amounts of training data. After training, each system's accuracy was recorded on the test set. All tests were run on the full test set. To reduce statistical fluctuations, the results of this process of dividing the examples, training, and testing were averaged over a number of runs (10 for the soybean data and 25 for the DNA data). Additionally, the random seeds for the back propagation algorithm were reset for each run. Training speed, testing speed, training set accuracy and test set accuracy were recorded for each run. Experiments were conducted with implementations of the following algorithms: back propagation, ID3, AQ (Michalski and Larson, 1983), perceptron, and Extentron. For more details on the specific implementations of ID3 and back propagation, see (Mooney et al., 1990). Statistical significance was measured using a Student t-test for paired difference of means at the 0.05 level of confidence (i.e. 95% certainty that the differences are not due to random chance).

We also tested the Extentron by repeating the standard "leave-one-out" (cross-validation) method performed by Towell et. al. on the DNA data set (Towell et al., 1990). This experiment proceeds by training with $N-1$ examples (where N is the size of the data set) and testing with the example left out. This process is repeated once for each example in the data set. The error rate is measured as the number of errors on the single test cases, divided by N. Again, to reduce the error due to statistical fluctuations the entire process was repeated 10 times and the results averaged.

4.2 Results

The experimental results are shown as learning curves in figures 4 and 5. Note that these graphs

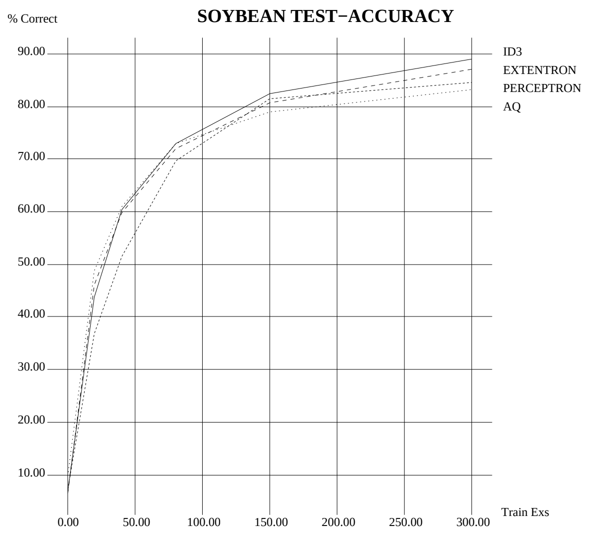

Figure 4: **Learning curves for the the soybean data set.** All curves were generated using 262 *novel* test examples. Significant differences were found between ID3 and Extentron for 150 and 300 training examples, between Extentron and perceptron for all but 150 training examples.

measure generalization over *novel* test data, as opposed to typical training data learning curves. These graphs show how the accuracy of each system changes as larger training sets are used. For the soybean data set, significant differences were found between ID3 and the Extentron, and between the Extentron and the simple perceptron (see caption, figure 4). For the DNA data set, back propagation and the Extentron performed equivalently, and each significantly out-performed the symbolic algorithms. Figure 6 compares the average training times of back propagation, ID3, and the Extentron.

Table 1 contains the number of errors for the "leave-one-out" test on the DNA examples. This table is identical to the one reported by Towell et. al., except for the inclusion of the Extentron results. In all cases, the algorithms listed correctly classify all members of the training sets. Towell et. al. report a statistical difference (99.95% certainty) between KBANN and back propagation.[1] Unfortunately, we did not have the data to test the significance between back propagation and the Extentron for this experiment, but we suspect the difference is not significant (since we found no significant differences in the DNA tests of figure 5).

5 Discussion of Results

As shown in figure 5 and table 1, the Extentron algorithm generalizes a well as back propagation, using only a fraction of the training time (see figure 6).

[1]They attribute this difference to the fact that KBANN starts with additional domain knowledge.

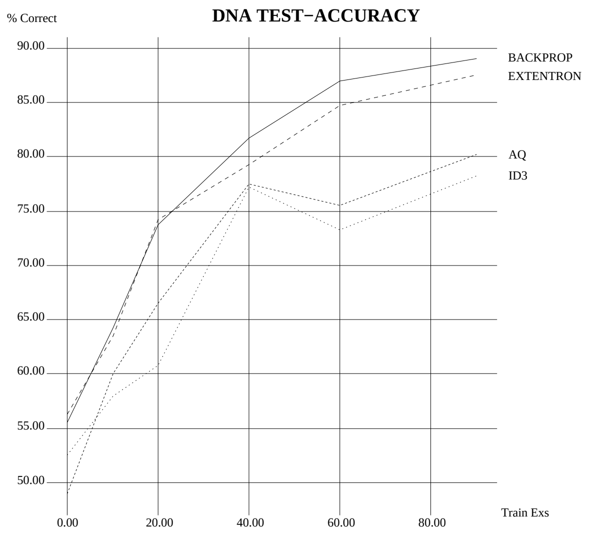

Figure 5: **Learning curves for the DNA data set.** All curves were generated using 16 *novel* test examples. No significant differences were found between back propagation and Extentron. Significant differences were found between each of the above and the two symbolic algorithms.

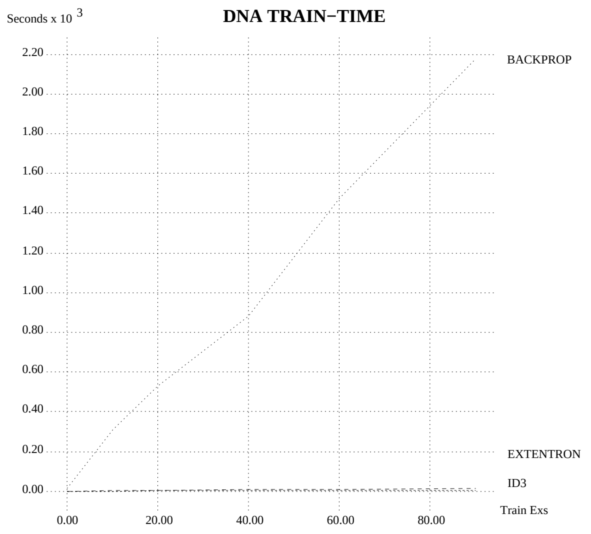

Figure 6: **Timing comparison of Backprop, Extentron and ID3.**

System	Error Rate
KBANN	4/106
Back Propagation	8/106
Extentron	9/106
O'Neill	12/106
Nearest Neighbor	13/106
ID3	19/106

Table 1: **DNA Error rates using leave-one-out method.**

The Extentron also performs better than the simple perceptron on the soybean data set, which was expected since the soybean data set is not linearly separable. In the DNA tests, the Extentron's superiority to ID3 and AQ duplicates the results reported by Fisher and McKusick (Fisher and McKusick, 1989) that neural networks are better at learning "N-of-M" concepts than symbolic algorithms.

The soybean tests and statistical results were run on a TI Explorer 1. To determine a good value for the accuracy parameter, the Extentron was trained with the entire soybean data set with accuracy set to 100%. After 6 layers, the Extentron consistently achieved 96% accuracy on the examples, and after 10 layers it converged. As a result, for the trials shown in figure 4, the Extentron was run with its accuracy parameter set at 0.95. This might account for the Extentron's inferior performance to ID3, since the Extentron was not forced to converge on the training data. Note that the Extentron never required more than three layers to complete any of the training runs on the soybean data set.

Unfortunately, the slowness of the back propagation algorithm prevented us from testing it using the full soybean data set.[2] However, in other tests using a subset of the soybean data, Mooney et. al. report nearly identical performance between the perceptron and back propagation. Since the Extentron proved to be superior to the perceptron, it is likely that the Extentron and back propagation perform similarly on this data set.

6 Future Work

There are many other experiments which could be performed to compare the Extentron with existing algorithms. To begin with, the Extentron should be run using a large problem such as NETtalk to further test its scalability.[3] It would also be interesting to compare the complexity of Extentron networks to other ontogenetic neural network algorithms, to see if the Extentron produces any savings in weights or neurons. Finally, it may be possible to generalize the Extentron to handle real-valued outputs, though we are skeptical that such attempts would succeed. During our development of the Extentron, we derived a closed-form solution for perceptron sum-squared error function. Though this solution could compute the optimal set of weights in one step, the work had to be abandoned because the weight vector yielding minimal error does *not* necessarily yield the best classification, even if the data is linearly separable.

[2]We were unable to get backprop to converge on any of the 10 test runs.

[3]Note that a simple perceptron can have *more* weights than a backprop net if the backprop net has a sufficiently small hidden layer.

7 Conclusion

In conclusion, it has been shown that the simple perceptron learning rule can be extended to solve difficult (not linearly separable) problems. Empirical results show that for naturally arising classification problems, the Extentron produces networks which generalize as well as back propagation in a fraction of the time.

References

Fisher, D. and McKusick, K. (1989). An experimental comparison of ID3 and backpropagation. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 788–793. San Mateo, CA: Morgan Kaufmann.

Hawley, D. and McClure, W. (1983). Compilation and analysis of Escherichia coli promoter DNA sequences. *Nucleic Acids Research*, 11:2237–2255.

Lang, K. J. and Witbrock, M. J. (1988). Learning to tell two spirals apart. *Proceedings of the 1988 Connectionist Models Summer School.*

Michalski, R. and Larson, J. (1983). Incremental generation of VL1 hypothesis: The underlying methodology and the description of the program AQ11. Technical Report ISG 83-5: University of Illinois at Urbana-Champaign.

Minsky, M. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press. Expanded edition.

Mooney, R., Shavlik, J. W., Towell, G., and Grove, A. (1990). An experimental comparison of symbolic and connectionist learning algorithms. pages 171–176.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Reinke, R. (1984). Knowledge acquisition and refinement tools for the advise meta-expert system. Master's thesis: University of Illinios at Urbana-Champaign.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, (65):386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 318–362. Cambridge, MA: MIT Press.

Shavlik, J. W., Mooney, R. J., and Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143.

Towell, G., Shavlik, J., and Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. volume 2, pages 861–866.

Utgoff, P. E. (1988). Perceptron trees: A case study in hybrid concept representations. *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 601–606.

Utgoff, P. E. and Brodley, C. E. (1990). An incremental method for finding multivariate splits for decision trees. *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 58–65.