Analysis of Decision Boundaries Generated by Constructive Neural Network Learning Algorithms

C-H Chen, R. G. Parekh, J. Yang, K. Balakrishnan, & V. Honavar *
Department of Computer Science
226 Atanasoff Hall,
Iowa State University,
Ames, IA 50011. U.S.A.

Abstract

Constructive learning algorithms offer an approach to incremental construction of near-minimal artificial neural networks for pattern classification. Examples of such algorithms include *Tower*, *Pyramid*, *Upstart*, and *Tiling* algorithms which construct multilayer networks of *threshold logic units* (or, multilayer perceptrons). These algorithms differ in terms of the topology of the networks that they construct which in turn biases the search for a decision boundary that correctly classifies the training set. This paper presents an analysis of such algorithms from a geometrical perspective. This analysis helps in a better characterization of the search bias employed by the different algorithms in relation to the geometrical distribution of examples in the training set. Simple experiments with non linearly separable training sets support the results of mathematical analysis of such algorithms. This suggests the possibility of designing more efficient constructive algorithms that dynamically choose among different biases to build near-minimal networks for pattern classification.

1 Introduction

Multi-layer networks of threshold logic units (TLU) or multi-layer perceptrons (MLP) offer a particularly attractive framework for the design of pattern classification systems for a number of reasons including: potential for parallelism and fault tolerance; significant representational and computational efficiency that they offer over disjunctive normal form (DNF) functions and decision trees [Gallant, 1993]; and simpler digital hardware realizations than their continuous counterparts. An output y_{jp} in response to an input pattern $\mathbf{X}_p = [x_{0p}\ x_{1p}\ \cdots\ x_{Np}]$ of a TLU or neuron j with a weight vector $\mathbf{W}_j = [w_{j0}\ w_{j1}\ \cdots\ w_{jN}]$ (where $\forall p\ x_{0p} = 1$ and $w_{j0} = T_j$ is the threshold for neuron j) is given by: $y_{jp} = 1$ if $\sum_{i=0}^{N} w_{ji} x_{ip} > 0$ and $y_{jp} = -1$ otherwise.

Such a TLU implements a $(N-1)$-dimensional hyperplane given by $\mathbf{W} \cdot \mathbf{X} = 0$ which partitions the N-dimensional Euclidian pattern space defined by the coordinates $x_1 \cdots x_N$ into two regions (or two classes). Such a neuron computes the bipolar hardlimiter function f of its net input. That is, if the net input $u_p = \mathbf{W}_j \cdot \mathbf{X}_p$, $f(u_p) = 1$ if $u_p > 0$ and $f(u_p) = -1$ otherwise. A given set of *examples* $S = S_+ \cup S_-$ where $S_+ = \{(\mathbf{X}_p, d_{jp}) \mid d_{jp} = 1\}$ and $S_- = \{(\mathbf{X}_p, d_{jp}) \mid d_{jp} = -1\}$ (where d_{jp} is the desired output of the pattern classifier for the input pattern $\mathbf{X}_p$), is said to be *linearly separable* if and only if $\exists \hat{\mathbf{W}}_j$ such that $\forall \mathbf{X}_p \in S_+$, $\hat{\mathbf{W}}_j \cdot \mathbf{X}_p > 0$ and $\forall \mathbf{X}_p \in S_-, \hat{\mathbf{W}}_j \cdot \mathbf{X}_p \leq 0$. A number of iterative algorithms are available for finding such a $\hat{\mathbf{W}}_j$ (if one exists — i.e., when S is linearly separable) (Nilsson, 1965). Most of them use some variant of the perceptron weight update rule: $\mathbf{W_j} \leftarrow \mathbf{W_j} + \eta(d_{jp} - y_{jp})\mathbf{X}_p$ (where η is the learning rate). However when S is not linearly separable, such algorithms behave poorly (i.e., the classification accuracy on the training set can fluctuate wildly from iteration to iteration). Several extensions to the perceptron weight update rule e.g., *pocket algorithm* (Gallant, 93), *thermal perceptron* (Frean, 1990), *loss minimization algorithm* (Hrycej, 1992) are designed to find a reasonably good weight vector that correctly classifies a large fraction of the

*This research was partially supported by the National Science Foundation grant IRI-909580 to Vasant Honavar.

training set S when S is not linearly separable and converge to zero classification error when S is linearly separable.

When S is not linearly separable, a multi-layer network of TLUs is needed to learn a complex decision boundary that correctly classifies all the training examples. Constructive learning algorithms (Honavar & Uhr, 1993) are designed to incrementally construct near-minimal networks of neurons (whose complexity as measured by the number of nodes, links etc.) is commensurate with the complexity of the classification problem implicitly specified by the training set. This paper focuses on constructive algorithms that build multi-layer perceptrons. These include tower, pyramid (Gallant, 93), upstart (Frean, 1990), and tiling (Mézard & Nadal, 89) algorithms. The rest of this paper is organized as follows: Section 2 provides an analysis of different constructive learning algorithms from a geometrical perspective. Section 3 presents experimental results on a simple 2-dimensional data set that confirm the results of analysis presented in section 2. Section 4 concludes with a summary and discussion of some future research directions.

2 Geometrical Analysis of Decision Boundaries

This section describes several algorithms for the construction of multi-layer networks of threshold neurons for pattern classification. They are all based on the idea of decomposition of the hard task of determining the necessary network topology and weights to two subtasks — addition of threshold neurons (one or a few at a time) alternating with an iterative modification of the weight vector for each added neuron (while keeping the rest of the network unchanged). This permits the use of simpler (and generally faster) algorithms for training individual threshold neurons. This paper focuses on the geometrical analysis of the working of tower, pyramid and upstart algorithms which incrementally build networks with topologies shown in Figure 1.

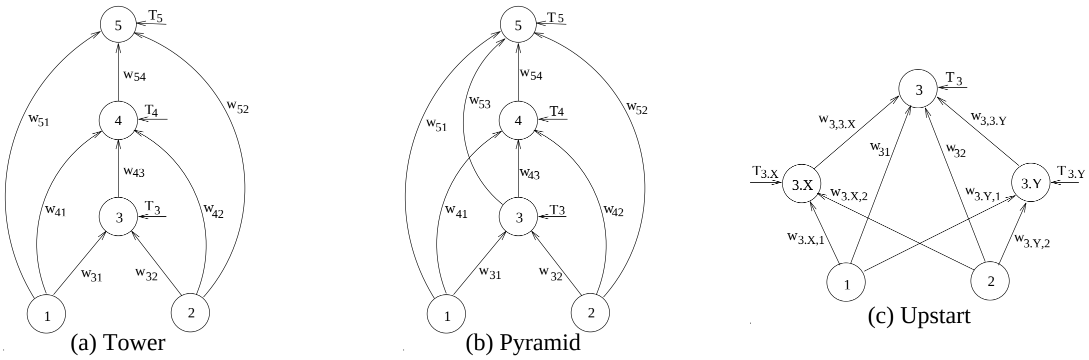

Figure 1: Topologies Generated by Tower, Pyramid and Upstart Algorithms

In the analysis that follows, we use the following notation: Let w_{ji} be the weight between neurons j and i, and T_j be the threshold for neuron j. Let H_j be the hyperplane given by $\mathbf{W}_j \cdot \mathbf{X} = \sum_1^N w_{ji}x_i + T_j = 0$. When the neuron j receives input from neurons other than the input neurons, its decision boundary gets modified. We denote this composite decision boundary by $B_j = 0$. We say that a particular pattern $\hat{\mathbf{X}}$ lies on the positive side of the hyperplane H_j if $\mathbf{W}_j \cdot \hat{\mathbf{X}} > 0$, or in shorthand (in the interest of minimizing notational clutter albeit at the cost of slight abuse of notation), we write $H_j > 0$ for all such patterns. We will use analogous notation when we talk about more complex decision hypersurfaces B_j.

2.1 The Tower Algorithm

The tower algorithm constructs a tower of TLUs. The bottom-most neuron in the tower receives n inputs, one for each component of the pattern vector. The kth neuron in the tower receives as inputs, each of the n components of the pattern as well as the output from the $(k-1)$th neuron immediately below it (see Figure 1(a)). The tower is built by training one neuron at a time until the desired classification accuracy is attained on the training set or addition of a neuron ceases to reduce the classification error.

The first unit trained is unit 3. The hyperplane it implements and the corresponding output are (see Figure 2(a)): $w_{31}x_1 + w_{32}x_2 + T_3 = 0$ and $y_3 = f(w_{31}x_1 + w_{32}x_2 + T_3)$ respectively. Suppose a new unit, 4, is added to the network and trained using the pocket (with ratchet) version of the perceptron algorithm. The composite decision boundary B_4 and the output are: $w_{41}x_1 + w_{42}x_2 + T_4 + w_{43}y_3 = 0$ and $y_4 = f(w_{41}x_1 + w_{42}x_2 + T_4 + w_{43}y_3)$ respectively. Without the influence of unit 3, the hyperplane H_4 for unit 4 is $w_{41}x_1 + w_{42}x_2 + T_4 = 0$. Assuming $w_{43} > 0$, there are 4 cases to consider:

1. $H_4 > 0$ and $y_3 = 1$: Since $w_{41}x_1 + w_{42}x_2 + T_4 > 0$ and $w_{43}y_3 > 0$ for all patterns on positive side of H_4 and H_3, $B_4 > 0$. That is, all the patterns that fall on the positive side of both H_4 and H_3 fall on the positive side of the hypersurface $B_4 = 0$.

2. $H_4 < 0$ and $y_3 = -1$: Clearly, since $w_{41}x_1 + w_{42}x_2 + T_4 < 0$ and $w_{43}y_3 < 0$, $B_4 < 0$. That is, all patterns that fall on the negative side of both H_4 and H_3, fall on the negative side of the hypersurface $B_4 = 0$.

3. $H_4 > 0$ and $y_3 = -1$: In this case, $w_{41}x_1 + w_{42}x_2 + T_4 + w_{43}y_3$ can be positive or negative based on the values of $w_{41}x_1 + w_{42}x_2 + T_4$ and $w_{43}y_3$. The composite boundary B_4 is obtained by shifting H_4 by the amount of $w_{43}y_3$.

4. $H_4 < 0$ and $y_3 = 1$: This case is exactly the same as case 3 but the sign of $w_{41}x_1 + w_{42}x_2 + T_4$ and $w_{43}y_3$ are the reversed.

Thus, the resulting decision boundary $B_4 = 0$ is shown in Figure 2(b). (For $w_{43} < 0$, the result will be a mirror image about H_4.)

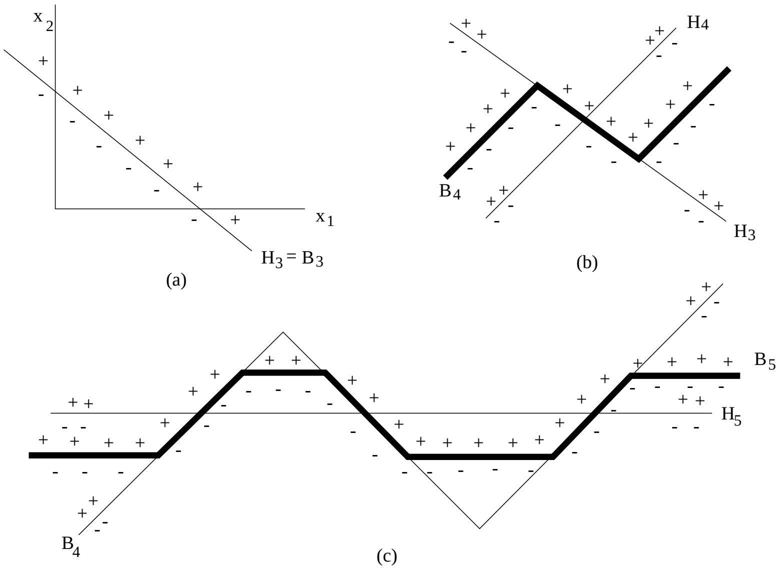

Figure 2: Decision Boundaries for the Tower Network

Now, suppose another unit, neuron 5, is added. The corresponding composite decision boundary B_5 is given by $w_{51}x_1 + w_{52}x_2 + T_5 + w_{54}y_4 = 0$. Disregarding the influence of unit 4, the hyperplane H_5 for unit 5 is $w_{51}x_1 + w_{52}x_2 + T_5 = 0$. Assuming $w_{53} > 0$, we need to consider 4 cases:

1. $H_5 > 0$ and $y_4 = 1$: Clearly, $B_5 > 0$ since $w_{51}x_1 + w_{52}x_2 + T_5 > 0$ and $w_{54}y_4 > 0$. So, all patterns that lie on positive side of both H_5 and B_4 fall on the positive side of B_5.

2. $H_5 < 0$ and $y_4 = -1$: Clearly, $B_5 < 0$ since $w_{51}x_1 + w_{52}x_2 + T_5 < 0$ and $w_{54}y_4 < 0$. So, all patterns that lie on the negative side of both H_5 and B_4 fall on the negative side of B_5.

3. $H_5 > 0$ and $y_4 = -1$: In this case, $w_{51}x_1 + w_{52}x_2 + T_5 + w_{54}y_4$ can be positive or negative based on the values of $w_{51}x_1 + w_{52}x_2 + T_5$ and $w_{54}y_4$. The composite boundary B_5 is essentially H_5 shifted by the amount corresponding to $w_{54}y_4$.

4. $\underline{H_5 < 0 \text{ and } y_4 = 1}$: This case is exactly the same as case 3 but the signs of $w_{51}x_1 + w_{52}x_2 + T_5$ and $w_{54}y_4$ are reversed.

Thus, the final decision boundary will be B_5 in Figure 2(c). (For $w_{54} < 0$, the result will be just a mirror image about H_5). The same procedure for analysis of decision boundaries can be applied to understand the consequences of successive addition of neurons.

2.2 The Pyramid Algorithm

The pyramid algorithm constructs a network of TLUs, much like the tower algorithm, training one neuron at a time, except that the kth neuron in the pyramid receives as input, each of the n components of the pattern vector, as well as the outputs from each of the $(k-1)$ neurons below it (See Figure 1(b)).

First we note that for the first neuron added (neuron 4), the decision boundary is same as in the case of the tower algorithm. Now, suppose the algorithm adds neuron 5. To analyze the resulting decision boundary $B_5 = 0$, unlike in the case of the tower algorithm, we now need to consider the hyperplanes $B_3 = H_3 = 0$ as well as $B_4 = 0$ and $H_5 = 0$. The composite decision boundary $B_5 = 0$ is $w_{51}x_1 + w_{52}x_2 + T_5 + w_{54}y_4 + w_{53}y_3 = 0$. Disregarding the influence of neurons 4 and 3, the hyperplane H_5 for neuron 5 is $w_{51}x_1 + w_{52}x_2 + T_5 = 0$. Assuming $w_{54} > 0$ and $w_{54} > w_{53}$, there are 6 cases:

1. $\underline{H_5 > 0,\ y_4 = 1 \text{ and } y_3 = \pm 1}$: Clearly, $B_5 > 0$ since $w_{51}x_1 + w_{52}x_2 + T_5 > 0$ and $|w_{54}y_4| > |w_{53}y_3|$.
2. $\underline{H_5 < 0,\ y_4 = -1 \text{ and } y_3 = \pm 1}$: Clearly, $B_5 < 0$ since $w_{51}x_1 + w_{52}x_2 + T_5 < 0$ and $|w_{54}y_4| > |w_{53}y_3|$.
3. $\underline{H_5 > 0,\ y_4 = -1 \text{ and } y_3 = 1}$: The composite boundary B_5 is obtained by shifting H_5 by the amount of $w_{54}y_4 + w_{53}y_3$.
4. $\underline{H_5 < 0,\ y_4 = 1 \text{ and } y_3 = -1}$: Same as 3 with the sign of the shift reversed.
5. $\underline{H_5 > 0,\ y_4 = -1 \text{ and } y_3 = -1}$: The composite boundary B_5 is obtained by shifting H_5 by the amount of $w_{54}y_4 + w_{53}y_3$.
6. $\underline{H_5 < 0,\ y_4 = 1 \text{ and } y_3 = 1}$: Same as 5 with the sign of the shift reversed.

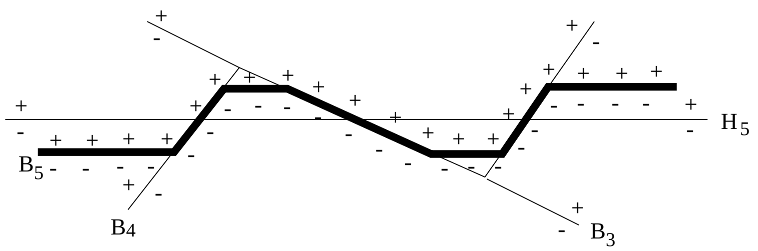

Figure 3: Decision Boundaries for the Pyramid Network

For all the other cases (corresponding to different signs of w_{54} and w_{53}), the analysis is quite similar. It is straightforward mathematically to carry out similar calculations for more complex decision boundaries that result from the addition of more neurons.

2.3 The Upstart Algorithm

The upstart algorithm starts by training a single TLU Z. If the training set S is linearly separable, Z correctly classifies every example in $S = S_+ \bigcup S_-$. Otherwise, the set of samples misclassified by Z can be grouped into two sets: the set M_+ of positive examples wrongly classified by Z as negative; and the set M_- of negative examples wrongly classified by Z as positive. Now two daughters X and Y are trained so as to correct the errors committed by Z. This is accomplished by training X on the training set $M_- \bigcup S_+$; and Y

on $M_+ \bigcup S_-$. The outputs of X and Y are then weighted by sufficiently large negative and positive weights respectively and fed as additional inputs to Z (see Figure 1(c)). [1]

In this case, we have $H_3 = B_3 = w_{31}x_1 + w_{32}x_2 + T_3 = 0$. Next, we add either an X daughter or a Y daughter of Z (or both) to the network. Suppose we add X daughter, and then Y daughter. Note that in this case, each neuron that is added changes the decision boundary corresponding to B_3. Suppose the X daughter and Y daughter of 3 are $3.X$ and $3.Y$ respectively. Let the weights between node 3 and its X and Y daughters be $w_{3,3.X}$ and $w_{3,3.Y}$ respectively, and the weights from input neuron i to $3.X$ and $3.Y$ be respectively $w_{3.X,i}$ and $w_{3.Y,i}$.

The composite decision boundary B_3 after generating X daughter is given by $B_3(new) = w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} = 0$. The algorithm requires $w_{3,3.X} < 0$. There are 4 cases to consider:

1. $\underline{B_3 > 0,\ y_{3.X} = -1}$: Clearly $w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} > 0$, and hence $B_3(new) > 0$.

2. $\underline{B_3 < 0,\ y_{3.X} = 1}$: Clearly $w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} < 0$, and hence $B_3(new) < 0$.

3. $\underline{B_3 < 0,\ y_{3.X} = -1}$: Here the boundary gets shifted in a manner analogous to that explained in the previous sections.

4. $\underline{B_3 > 0,\ y_{3.X} = 1}$: Same as 3.

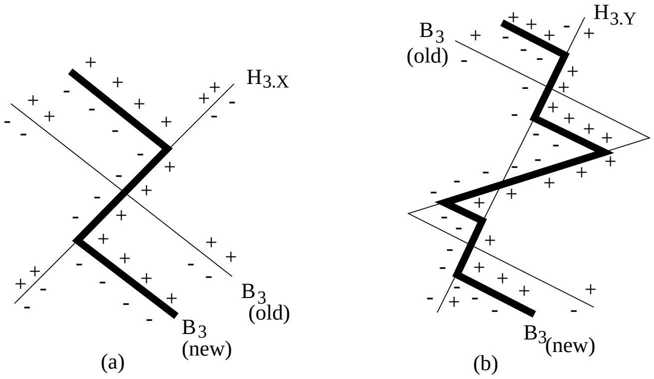

Figure 4: Decision Boundaries for the Upstart Network

The new decision boundary is shown in Figure 4(a). Now, suppose we add Y daughter thereby changing the decision boundary further. The resulting decision surface is given by: $B_3(new) = w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} + w_{3,3.Y}y_{3.Y} = 0$. As per the algorithm, $w_{3,3.Y} > 0$. Then possible cases are:

1. $\underline{B_3 > 0,\ y_{3.Y} = 1}$: Clearly $w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} + w_{3,3.Y}y_{3.Y} > 0$, and hence $B_3(new) > 0$.

2. $\underline{B_3 < 0,\ y_{3.Y} = -1}$: Clearly $w_{31}x_1 + w_{32}x_2 + T_3 + w_{3,3.X}y_{3.X} + w_{3,3.Y}y_{3.Y} < 0$, and hence $B_3(new) < 0$.

3. $\underline{B_3 < 0,\ y_{3.Y} = 1 \text{ or } B_3 > 0,\ y_{3.Y} = -1}$: The boundary gets shifted for reasons similar to those encountered before (See Figure 4(b)).

As new daughter neurons get added, similar analysis can be used to determine the resulting decision boundaries mathematically.

3 Experiments

In this section we present experimental results verifying the analysis of decision boundaries presented in Section 2. Since our primary concern in this paper is with understanding the detailed working of the various algorithms that were considered, the experiments were limited to the exclusive OR (XOR) problem (a non

[1] The upstart algorithm (Frean, 1990) works with binary (as opposed to bipolar input and output values). To facilitate direct comparison, we assume that the weights and thresholds are transformed so as to implement an equivalent bipolar mapping (Gallant, 93) after training the network with binary inputs and outputs.

linearly separable data set with 4 data points) so that the resulting decision boundaries could be easily represented visually. Each TLU was trained using the *fixed correction perceptron* algorithm with *ratchet modification* (Gallant, 93). Fig. 5 shows the evolution of the decision boundaries for a tower network when each new TLU was trained for 100 samples randomly chosen (with replacement) from the data set. The training was successful after addition of 2 TLUs resulting in a network that is similar to the network shown in Fig. 1 but with just 4 units. The tower network needed 4 units to correctly classify all patterns. The first graph in Fig. 5 depicts the hyperplane H_3 corresponding to unit 3 of Fig. 1. For the run shown, the corresponding weights are $w_{31} = 0.8193$, $w_{32} = 1.4898$, and $T_3 = 1.3054$. The second graph shows the hyperplane H_3, the hyperplane H_4 (due to unit 4 disregarding the influence of the output from unit 3) and the composite decision boundary corresponding to $B_4 = 0$. The weights for unit 4 are $w_{41} = -1.2882$, $w_{42} = -0.8713$, $w_{43} = 3.7282$, and $T_4 = -2.37531$.

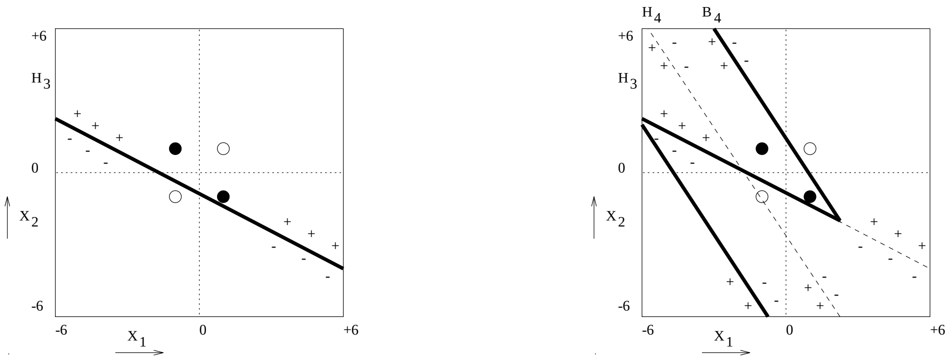

Figure 5: Decision Boundaries for the Tower Network

The decision boundaries can be understood in terms of the analysis presented in the previous section.

1. $H_4 > 0$ and $y_3 = 1$: B_4 becomes $-1.2882x_1 - 0.8713x_2 - 2.3753 + 3.72815(1) = 0$. For all points (x_1, x_2) where $H_4 > 0$, clearly $B_4 > 0$ and all points in this region are positive.

2. $H_4 < 0$ and $y_3 = -1$: B_4 becomes $-1.2882x_1 - 0.8713x_2 - 2.3753 + 3.72815(-1) = 0$. For all points (x_1, x_2) where $H_4 < 0$, clearly $B_4 < 0$ and all points in this region are negative.

3. $H_4 > 0$ and $y_3 = -1$: B_4 becomes $-1.2882x_1 - 0.8713x_2 - 2.3753 + 3.72815(-1) = 0$. Some of the points that were declared as positive by H_4 are now classified as negative by B_4. This is represented by an upward shift of the decision boundary so that B_4 is parallel to H_4 in this region.

4. $H_4 < 0$ and $y_3 = 1$: B_4 becomes $-1.2882x_1 - 0.8713x_2 - 2.3753 + 3.72815(1) = 0$. Some of the points that were declared as negative by H_4 are now classified as positive by B_4. This is represented by a downward shift of the decision boundary so that B_4 is parallel to H_4 in this region.

Thus the experiments confirm the Z-shaped decision boundary for a tower network with 4 units. Results of similar experiments with pyramid and upstart algorithms also agree with the theoretical conclusions drawn in the previous section.

Next we analyzed the decision boundaries generated by the tower, pyramid, and upstart algorithms using the XOR data set but this time allowing each new TLU just 30 epochs to train. Limiting the time allowed for the units to train simulates (albeit crudely) the performance of the algorithm on a data set that requires a more complex decision boundary. The evolution of the composite decision boundary after the addition of a new TLU for the tower, pyramid, and upstart algorithms is depicted in Figs. 6, 7, and 8 respectively. A case-by-case analysis (as described in Section 2) can be conducted in each case to explain the generation of the decision boundary. Owing to space constraints, we omit the details.

The tower network needed 4 units besides the input units to correctly classify all patterns. In the tower algorithm each unit receives information only from the input units and the output of the most recently trained unit. Dearth of information (as compared to the pyramid and upstart algorithms) coupled with insufficient training time for each unit is a possible explanation for the inability of the tower network to find the desired decision boundary quickly.

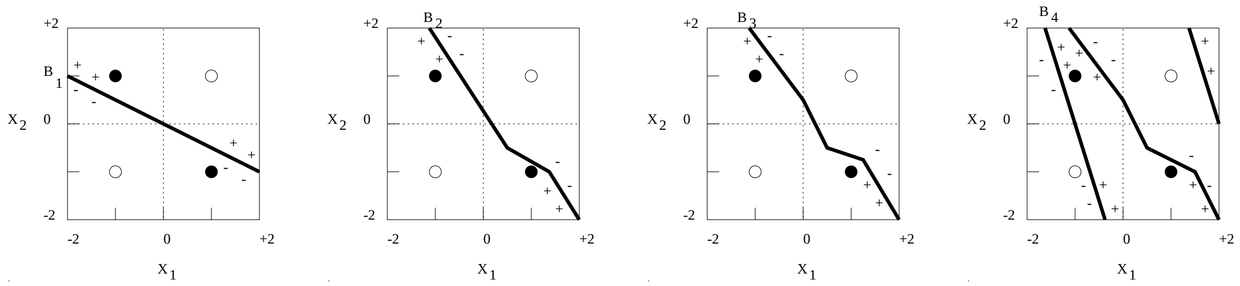

Figure 6: Decision Boundaries for the Tower Network

In the pyramid algorithm, each new unit has access to the outputs from the previous units along with the information from the inputs. This is possibly an advantage over the tower network and because it enables the pyramid network to generate more complex decision boundaries using a fewer number of units. This might explain the fact that the pyramid network generated in this experiment contains just 3 units besides the input units.

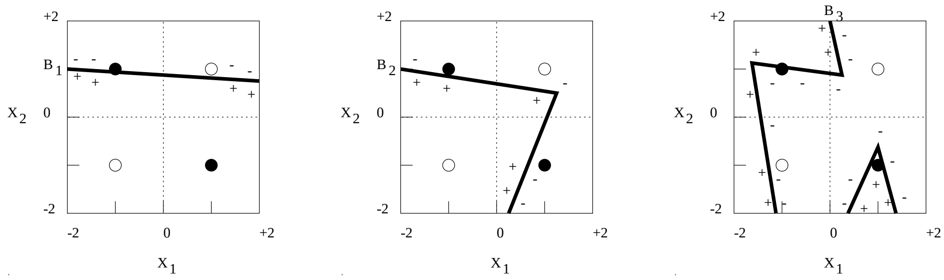

Figure 7: Decision Boundaries for the Pyramid Network

In the upstart algorithm, each unit is trained with just the information from the input patterns. However, the construction of daughter units is designed specifically to correct certain types of errors made by the mother unit. The training set for the daughter units is explicitly constructed to ensure that after being trained each daughter would serve to reduce the the number of mis-classifications made by the mother. Since the upstart algorithm focuses its search effort at each step on correcting specific types of errors, it can potentially converge using a lesser number of units than the tower and pyramid algorithms. In the experiment with the XOR data set the upstart algorithm converged with just 2 units (an output unit and a Y-daughter) besides the input units. The advantage of the upstart algorithm in terms of the nature of decision boundaries constructed is not obvious from toy data sets such as the XOR. However, it seems that like the pyramid algorithm the upstart algorithm is capable of generating more complex decision boundaries using a fewer number of units.

4 Summary and Conclusions

In this paper we have presented an approach for the analysis of decision boundaries generated by several constructive neural network learning algorithms. All such algorithms attempt to construct increasingly complex decision surfaces by composing two or more simpler surfaces and (at least in the case of the algorithms examined in this paper) in theory are capable of constructing a network of threshold neurons that correctly classifies a given training set. However, their performance in practice is likely to be tied closely to the particular design choices that they embody (such as where to add a new neuron, how to connect it with the existing neurons, etc.). As shown by our analysis of tower, pyramid and upstart algorithms, these decisions

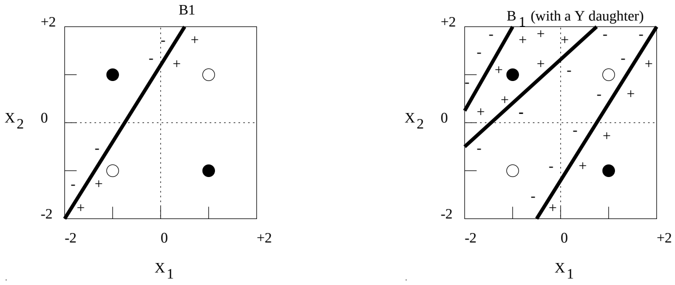

Figure 8: Decision Boundaries for the Upstart Network

determine how each algorithm partitions into regions so as to separate the positive samples from negative samples. In the case of these algorithms, since each neuron is trained independently (i.e., one at a time), the composite decision boundary that results from the addition of a neuron is made of segments of the decision boundary realized by the network prior to the addition of the new unit and segments parallel to the hyperplane that corresponds to the newly added neuron (disregarding the influence of any unit in the network other than the input neurons). Although our experiments were restricted to 2-dimensional toy data sets such as the XOR (to facilitate visualization and easy verification of the analysis), the mathematical framework and the results of the analysis carry over to more complex problems. A goal of this research is a precise characterization of the inductive and representational biases that are implicitly employed by a range of closely related constructive neural network learning algorithms. This would identify properties of algorithms that would help them perform well on data sets that share certain properties (in terms of the distribution of patterns in the pattern space) and to design more powerful algorithms of this class that can dynamically choose among different network construction strategies. Related work in progress is aimed at empirical comparison of performance of various network algorithms (along with different weight modification procedures) on a number of artificial as well as real-world data sets.

References

Frean, M. (1990). Small nets and short paths: Optimizing neural computation. Ph.D. Thesis. Center for Cognitive Science. University of Edinburgh, UK.

Gallant, S. I. (1993). *Neural Network Learning and Expert Systems.* Cambridge, MA: MIT Press.

Honavar, V. (1990). Generative Learning Structures and Processes for Generalized Connectionist Networks. Ph.D. Thesis. University of Wisconsin, Madison, U.S.A.

Honavar, V., and Uhr, L. (1993). Generative Learning Structures and Processes for Connectionist Networks. *Information Sciences* 70, 75-108.

Hrycej, T. (1992). Modular Neural Networks. New York: Wiley.

Mézard, M., and Nadal, J. (1989). Learning in feed-forward networks: The tiling algorithm. *J. Phys. A: Math. and Gen.* 22. 2191-2203.

Nilsson, N. (1965). *Learning Machines.* New York: McGraw-Hill.