Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification

TR #97-06
Rajesh Parekh, Jihoon Yang, and Vasant Honavar
March 15, 1997

ACM Computing Classification System Categories (1991):
I.2.6 [*Artificial Intelligence*] Learning — connectionism and neural nets; I.5.1 [*Pattern Recognition*] Models — Neural nets.

Keywords:
neural networks, constructive learning algorithms, pattern classification, machine learning, supervised learning

Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa Sate University
Ames, Iowa 50011-1040. USA.

Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification

Rajesh Parekh, Jihoon Yang, and Vasant Honavar *
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, IA 50011. U.S.A.
{parekh|yang|honavar}@cs.iastate.edu

March 15, 1997

Abstract

Constructive learning algorithms offer an attractive approach for incremental construction of potentially near-minimal neural network architectures for pattern classification tasks. These algorithms help overcome the need for ad-hoc and often inappropriate choice of network topology in the use of algorithms that search for a suitable weight setting in an a-priori fixed network architecture. Several such algorithms proposed in the literature have been shown to converge to zero classification errors (under certain assumptions) on finite, non-contradictory training sets in 2-category classification tasks. The convergence proofs for each of these algorithms (with the exception of the *Upstart* and *Perceptron Cascade*) rely on the assumption that the pattern attributes are either binary or bipolar valued. This paper explores multi-category extensions of several constructive neural network learning algorithms for classification tasks where the input patterns may take on real-valued attributes. In each case, we establish the convergence to zero classification errors on a multi-category classification task. Results of experiments with non-linearly separable multi-category datasets demonstrate the feasibility of this approach and suggest several interesting directions for future research.

*This research was partially supported by the National Science Foundation grants IRI-9409580 and IRI-9643299 to Vasant Honavar.

1 Introduction

Multi-layer networks of threshold logic units (also called threshold neurons or TLU) or multi-layer perceptrons (MLP) offer an attractive framework for the design of pattern classification and inductive knowledge acquisition 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.

1.1 Threshold Logic Units

A single TLU, also known as *perceptron*, can be trained to classify a set of input patterns into one of two classes. A TLU is an elementary processing neuron that computes a hard-limiting function of the weighted sum of its inputs. Assuming that the patterns are drawn from an N-dimensional Euclidean space, the output O^p, of a TLU with weight vector $\mathbf{W}$, in response to a pattern $\mathbf{X}^p$ is

$$\begin{aligned} O^p &= 1 \text{ if } \mathbf{W} \cdot \mathbf{X}^p \geq 0 \\ &= -1 \text{ otherwise} \end{aligned}$$

A TLU that implements the bipolar hard-limiting functions (i.e., the TLU's outputs are 1 and -1) is called a *bipolar* TLU as against the TLU that implements the binary hard-limiting function (with outputs 1 and 0) which is referred to as a *binary* TLU. Unless explicitly stated, we will work with bipolar TLUs. A TLU implements a $(N-1)$-dimensional hyperplane given by $\mathbf{W} \cdot \mathbf{X} = 0$ which partitions the N-dimensional Euclidean pattern space defined by the coordinates $x_1, \cdots, x_N$ into two regions (or two classes). Given a set of *examples* $S = S^+ \cup S^-$ where $S^+ = \{(\mathbf{X}^p, C^p) \,|\, C^p = 1\}$ and $S^- = \{(\mathbf{X}^p, C^p) \,|\, C^p = -1\}$ (C^p is the desired output of the pattern classifier for the input pattern $\mathbf{X}^p$), it is the goal of a *perceptron training* algorithm to attempt to find a weight vector $\hat{\mathbf{W}}$ such that $\forall \mathbf{X}^p \in S^+$, $\hat{\mathbf{W}} \cdot \mathbf{X}^p \geq 0$ and $\forall \mathbf{X}^p \in S^-$, $\hat{\mathbf{W}} \cdot \mathbf{X}^p < 0$. If such a weight vector ($\hat{\mathbf{W}}$) exists for the pattern set S then S is said to be *linearly separable*. Several iterative algorithms are available for finding such a $\hat{\mathbf{W}}$ if one exists [Rosenblatt, 1958; Minsky & Pappert, 1969; Nilsson, 1965; Duda & Hart, 1973]. Most of these are variants of the *perceptron weight update rule*: $\mathbf{W} \leftarrow \mathbf{W} + \eta(C^p - O^p)\mathbf{X}^p$ (where $\eta > 0$ 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* with *ratchet modification* [Gallant, 1990], *Thermal perceptron algorithm* [Frean, 1990a; Frean, 1992], *Loss minimization algorithm* [Hrycej, 1992], and the *Barycentric correction procedure* [Poulard, 1995] are designed to find a reasonably good weight vector that correctly classifies a large fraction of the training set S when S is not linearly separable and to converge to zero classification errors when S is linearly separable. For a detailed

comparison of the algorithms for training TLUs see [Yang *et al.*, 1997b]. Recently [Siu *et al.*, 1995] have established the necessary and sufficient conditions for a training set S to be non-linearly separable. They have also shown that the problem of identifying a largest linearly separable subset S_{Sep} of S is NP-complete. It is widely conjectured that no polynomial time algorithms exist for NP-complete problems [Garey & Johnson, 1979]. Thus, we rely on heuristic algorithms such as the *Pocket algorithm* with *ratchet modification* to correctly classify as large a subset of training patterns as possible within the given constraints (such as limited training time).

1.2 Constructive Learning Algorithms

When S is not linearly separable, a multi-layer network of TLUs is needed to learn a non-linear decision boundary that correctly classifies all the training examples. The focus of this paper is on *constructive* or *generative* learning algorithms that incrementally construct networks of threshold neurons to correctly classify a given (typically non-linearly separable) pattern set. Some of the motivations for studying such algorithms [Honavar, 1990; Honavar & Uhr, 1993] include:

- *Limitations of learning by weight modification alone within an a-priori fixed network topology:* Weight modification algorithms typically search for a solution weight vector that satisfies some desired performance criterion (e.g., classification error). In order for this approach to be successful, such a solution must lie within the weight-space being searched, and the search procedure employed must in fact, be able to locate it. This means that unless the user has adequate problem-specific knowledge that could be brought to bear upon the task of choosing an adequate network topology, the process is reduced to one of trial and error. Constructive algorithms can potentially offer a way around this problem by extending the search for a solution, in a controlled fashion, to the space of network topologies.

- *Complexity of the network should match the intrinsic complexity of the classification task:* It is desirable that a learning algorithm construct networks whose complexity (in terms of relevant criteria such as number of nodes, number of links, connectivity, etc.) is commensurate with the intrinsic complexity of the classification task (implicitly specified by the training data). Smaller networks yield efficient hardware implementations. Everything else being equal, the more compact the network, the more likely it is that it exhibits better generalization properties. Constructive algorithms can potentially discover near-minimal networks for correct classification of a given dataset.

- *Estimation of expected case complexity of pattern classification tasks:* Many pattern classification tasks are known to be computationally hard. However, little is known about the *expected* case complexity of classification tasks that are encountered, and successfully solved, by living systems - primarily because it is difficult to

mathematically characterize the properties of such problem instances. Constructive algorithms, if successful, can provide useful empirical estimates of expected case complexity of real-world pattern classification tasks.

- *Trade-offs among performance measures:* Different constructive learning algorithms offer natural means of trading off certain performance measures (like learning time) against others (like network size and generalization accuracy).

- *Incorporation of prior knowledge:* Constructive algorithms provide a natural framework for incorporating problem-specific knowledge into the initial network configuration and augmenting the network to encompass additional information from the new examples seen.

A number of algorithms that incrementally construct networks of threshold neurons for 2-category pattern classification tasks have been proposed in the literature. These include the *Tower*, *Pyramid* [Gallant, 1990], *Tiling* [Mézard & Nadal, 1989], *Upstart* [Frean, 1990b], *Perceptron Cascade* [Burgess, 1994], and *Sequential* [Marchand *et al.*, 1990]. With the exception of the *Sequential* learning algorithm, constructive learning algorithms are based on the idea of transforming the task of determining the necessary network topology and weights to two subtasks:

- Incremental addition of one or more threshold neurons to the network when the existing network topology fails to achieve the desired classification accuracy on the training set.

- Training the added threshold neuron(s) using some variant of the perceptron training algorithm.

In the case of the *Sequential* learning algorithm, hidden neurons are added and trained by an appropriate weight training rule to exclude as many patterns belonging to the same class as possible from the currently unexcluded patterns. The constructive algorithms differ in terms of their choices regarding: restrictions on input representation (e.g., binary, bipolar, or real-valued inputs); when to add a neuron; where to add a neuron; connectivity of the added neuron; weight initialization for the added neuron; how to train the added neuron (or a subnetwork affected by the addition); and so on. The interested reader is referred to [Chen *et al.*, 1995] for an analysis (in geometrical terms) of the decision boundaries generated by some of these constructive learning algorithms. Each of these algorithms can be shown to converge to networks which yield zero classification errors on any given training set wherein the patterns belong to one of two classes (i.e., 2-category classification). The convergence proof of the *Sequential* learning algorithm is based on the ability of the TLU weight training algorithm to exclude at least one formerly unexcluded pattern from the training set each time a new hidden neuron is trained. In the case of the other algorithms the convergence proof is based on the ability of the TLU weight training algorithm to find a weight setting for each newly added neuron or neurons such that the

number of pattern misclassifications is reduced by at least one each time a neuron (or a set of neurons) is added and trained and the network's outputs are recomputed. We will refer to such a TLU weight training algorithm as $\mathcal{A}$ and assume that it will correspond to an appropriate choice depending on the constructive algorithm being considered. In practice, the performance of the constructive algorithm depends partly on the choice of $\mathcal{A}$ and its ability to find weight settings that reduce the total number of misclassifications (or to exclude at least one formerly unexcluded pattern from the training set) each time new neurons are added to the network and trained. Some possible choices for $\mathcal{A}$ when the desired task is to maximize classification accuracy are the *Pocket algorithm* with *ratchet modification*, the *Thermal perceptron algorithm*, and the *Barycentric correction procedure*. A variant of the *Barycentric correction procedure* can be used to efficiently exclude patterns as desired by the *Sequential* learning algorithm.

1.3 Issues in Practical Pattern Classification Tasks

Pattern classification tasks often require assigning patterns to one of M ($M > 2$) classes. Although in principle, an M-category classification task can be reduced to an equivalent set of M 2-category classification tasks (each with its own training set constructed from the given M-category training set), a better approach might be one that takes into account the inter-relationships between the M output classes. For instance, the knowledge of membership of a pattern $\mathbf{X}^p$ in category Ψ_i can be used by the learning algorithm to effectively rule out its membership in a different category Ψ_j ($j \neq i$) and any internal representations learned in inducing the structure of Ψ_i can therefore be exploited in inducing the structure of some other category Ψ_j ($j \neq i$). In the case of most constructive learning algorithms, extensions to multiple output classes have not been explored. In other cases, only some preliminary ideas (not supported by detailed theoretical or experimental analysis) for possible multi-category extensions of 2-category algorithms are available in the literature. A preliminary analysis of the extension of constructive learning algorithms to handle multi-category classification tasks is presented in [Parekh *et al.*, 1995].

For pattern sets that involve multiple output classes, training can be performed either *independently* or by means of the *winner take all* (WTA) strategy. In the former, each output neuron is trained independently of the others using one of the TLU weight training algorithms mentioned earlier. The fact that the membership of a pattern in one class precludes its membership in any other class can be exploited to compute the outputs using the WTA strategy wherein, for any pattern, the output neuron with the highest net input is assigned an output of 1 and all other neurons are assigned outputs of -1. In the case of a tie for the highest net input all neurons are assigned an output of -1, thereby rendering the pattern incorrectly classified. The WTA strategy succeeds in correctly classifying patterns belonging to multiple output classes that are only pairwise separable from each other whereas the traditional method of computing the output of each neuron independently succeeds in correctly classifying all patterns only if the pat-

terns belong to classes that are linearly separable from each other [Gallant, 1993]. It is thus of interest to apply the WTA strategy for computing the outputs in constructive learning algorithms. For details on the adaptation of the TLU training algorithms to the WTA strategy see [Yang *et al.*, 1997b].

Additionally, practical classification tasks often involve patterns with real-valued attributes. The TLU weight training algorithms like the *Pocket algorithm* with *ratchet modification*, *Thermal perceptron algorithm*, and *Barycentric correction procedure* do handle patterns with real-valued attributes. However, extensions of the constructive learning algorithms to handle patterns with real-valued attributes have only been studied for the *Upstart* [Saffery & Thornton, 1991] and the *Perceptron Cascade* [Burgess, 1994] algorithms.

1.4 Notation

In this section we describe the pre-processing of the pattern set that is required to guarantee the convergence of certain multi-category constructive algorithms and outline the notation that will be used in the description of the algorithms and in the convergence proofs.

In the case of the *Tower*, *Pyramid*, *Upstart*, and *Perceptron Cascade* algorithms the pattern set must be preprocessed in order to guarantee convergence on real valued datasets. The preprocessing involves a projection of the individual patterns onto a parabolic surface. This projection is achieved by simply appending an additional attribute to each pattern. This attribute takes on a value equal to the sum of squares of the values of all other attributes in the pattern. Although the *Tiling* and the *Sequential* algorithms do not need the projection of the pattern set to guarantee convergence, the reader must note that such a projection would not hamper the convergence properties of these two algorithms.

The following notation is used through out the paper.
Output categories: $\Psi_1, \Psi_2, \ldots, \Psi_M$ (assuming $M > 2$)
Number of pattern attributes: N
Number of output neurons (equal to the number of categories): M
Input layer index: I
Indices for other layers (hidden and output): 1, 2, $\cdots$, L
Number of neurons in layer A: U_A
Indexing for neurons of layer A: $A_1, A_2, \ldots, A_{U_A}$
Threshold (or bias) for neuron i of layer A: $W_{A_i,0}$
Connection weight between neuron i of layer A and neuron j of layer B: W_{A_i,B_j}
Augmented pattern p: $\mathbf{X}^p = < X_0^p, X_1^p, \ldots, X_N^p >$, $X_0^p = 1$ for all p,
and $X_i^p \in \mathcal{R}$ for all p, i

Projected pattern p: $\hat{\mathbf{X}}^p = < X_0^p, X_1^p, \ldots, X_N^p, X_{N+1}^p >$,
$X_{N+1}^p = \sum_{i=0}^{N} (X_i^p)^2$
Net input for neuron A_j in response to pattern $\mathbf{X}^p$: $n_{A_j}^p$
Target output for pattern $\mathbf{X}^p$: $\mathbf{C}^p = < C_1^p, C_2^p, \ldots, C_M^p >$,
$C_i^p = 1$ if $\mathbf{X}^p \in \Psi_i$ and $C_i^p = -1$ otherwise
Observed output for pattern $\mathbf{X}^p$ at layer A: $\mathbf{O}_A^p = < O_{A_1}^p, O_{A_2}^p, \ldots, O_{A_k}^p >$ where $k = U_A$
Number of patterns wrongly classified at layer A: e_A

A pattern is said to be correctly classified at layer A when $\mathbf{C}^p = \mathbf{O}_A^p$. A function *sgn* is defined as $sgn(x) = -1$ if $x < 0$ and $sgn(x) = 1$ if $x \geq 0$ where $x \in \mathcal{R}$. We see that bipolar TLUs implement the *sgn* function. As is standard in neural networks literature we will assume that the input layer I neurons are *linear* neurons with a single input (whose weight is set to 1) and the output equal to the input. Thus, for an N-dimensional pattern set the input layer would have N linear neurons (one for each element of the pattern vector). Similarly, for projected pattern sets the input layer would have $N + 1$ linear neurons. Layers 1 through L have threshold neurons. In the figures of the networks constructed by the algorithms, the threshold (or bias) of each TLU is represented by a separate arrow attached to the respective neuron.

Against this background, the focus of this paper is on provably convergent multi-category learning algorithms for construction of networks of threshold neurons for pattern classification tasks with real-valued attributes. The rest of the paper is organized as follows: Sections 2 through 7 explore the *Tower*, *Pyramid*, *Upstart*, *Perceptron Cascade*, *Tiling* and *Sequential* learning algorithms respectively. In each case, convergence to zero classification errors is established for both the independent and WTA output strategies. Section 8 presents preliminary results on several artificial and real-world classification tasks. Section 9 concludes with a summary and a discussion of future research directions.

2 Tower Algorithm

The 2-category *Tower* algorithm [Gallant, 1990] constructs a tower of TLUs. The bottom-most neuron receives inputs from each of the N input neurons. The tower is built by successively adding neurons to the network and training them using $\mathcal{A}$ until the desired classification accuracy is achieved. Each newly added neuron receives input from each of the N input neurons and the output of the neuron immediately below itself and takes over the role of the network's output.

To handle patterns with real valued attributes it is necessary to consider the projection of the patterns onto a parabolic surface. The extension of the 2-category *Tower* algorithm to deal with multiple (M) output categories is accomplished by simply adding M neurons each time a new layer is added to the tower. Each neuron in the newly added

layer (which then serves as the network's output layer) receives inputs from the $N+1$ input neurons as well as the M neurons in the preceding layer. The resulting *Tower* network is shown in Fig. 1.

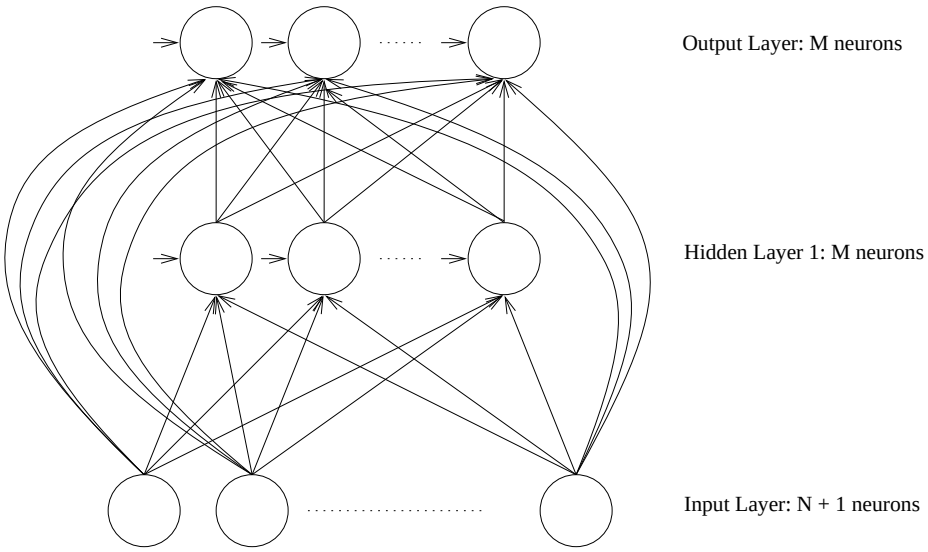

Figure 1: Tower Network

2.1 Multi-Category Tower Algorithm

1. Set the current output layer index $L = 0$.
2. Repeat the following steps until the desired training accuracy is achieved or the maximum number of hidden layers allowed is exceeded.
   (a) $L = L + 1$. Add M output neurons to the network at layer L. This forms the new output layer of the network. Connect each neuron in layer L to the $N+1$ input neurons and to each neuron in the preceding layer, $L-1$, if one exists.
   (b) Train the weights associated with neurons in layer L (the rest of the weights in the network are left unchanged).

2.2 Convergence Proof

Theorem 1:
There exists a weight setting for neurons in the newly added layer L in the multi-category *Tower* network such that the number of patterns misclassified by the network with L layers is less than the number of patterns misclassified prior to the addition of the L^{th} layer (i.e., $\forall L > 1,\ e_L < e_{L-1}$).

Proof:

Define $\kappa = \max_{p,q} \sum_{i=1}^{N} (X_i^p - X_i^q)^2$. For each pattern $\hat{\mathbf{X}}^p$, define ϵ_p as $0 < \epsilon_p < \min_{p,q \neq p} \sum_{i=1}^{N} (X_i^p - X_i^q)^2$. It is clear that $0 < \epsilon_p < \kappa$ for all patterns $\hat{\mathbf{X}}^p$. Assume that a pattern $\hat{\mathbf{X}}^p$ was not correctly classified at layer $L-1$ (i.e., $\mathbf{C}^p \neq \mathbf{O}_{L-1}^p$). Consider the following weight setting for the output neuron L_j ($j = 1 \ldots M$).

$$\begin{aligned} W_{L_j,0} &= C_j^p(\kappa + \epsilon_p - \sum_{i=1}^{N} (X_i^p)^2) \\ W_{L_j,I_i} &= 2C_j^p X_i^p \text{ for } i = 1 \ldots N \\ W_{L_j,I_{N+1}} &= -C_j^p \\ W_{L_j,L-1_j} &= \kappa \\ W_{L_j,L-1_k} &= 0 \text{ for } k = 1 \ldots M, k \neq j \end{aligned} \tag{1}$$

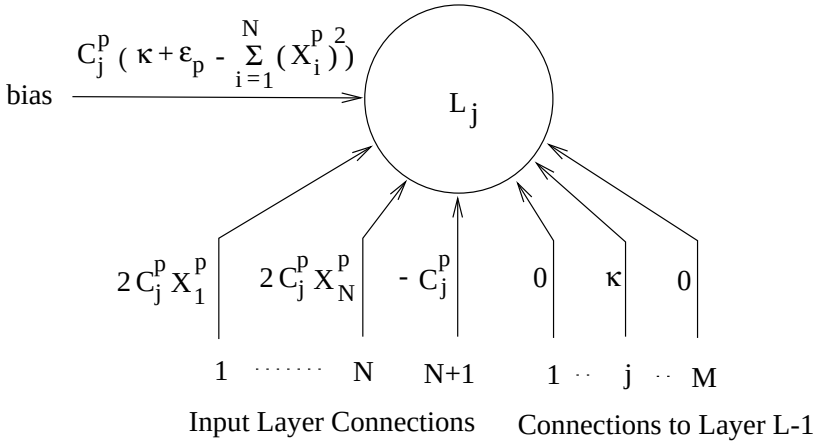

Figure 2: Weight setting for output neuron L_j in the Tower Network

For the pattern $\hat{\mathbf{X}}^p$ the net input $n_{L_j}^p$ of neuron L_j is:

$$\begin{aligned} n_{L_j}^p &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^p + \sum_{i=1}^{M} W_{L_j,L-1_i} O_{L-1_i}^p \\ &= C_j^p(\kappa + \epsilon_p - \sum_{i=1}^{N} (X_i^p)^2) + 2C_j^p \sum_{i=1}^{N} (X_i^p)^2 - C_j^p \sum_{i=1}^{N} (X_i^p)^2 + \kappa O_{L-1_j}^p \\ &= C_j^p(\kappa + \epsilon_p) + \kappa O_{L-1_j}^p \end{aligned} \tag{2}$$

If $C_j^p = -O_{L-1_j}^p$:

$$\begin{aligned} n_{L_j}^p &= C_j^p \epsilon_p \\ O_{L_j}^p &= sgn(n_{L_j}^p) \\ &= C_j^p \text{ since } \epsilon_p > 0 \end{aligned}$$

If $C_j^p = O_{L-1_j}^p$:

$$\begin{aligned} n_{L_j}^p &= (2\kappa + \epsilon_p) C_j^p \\ O_{L_j}^p &= sgn(n_{L_j}^p) \\ &= C_j^p \text{ since } \kappa, \epsilon_p > 0 \end{aligned}$$

Thus we have shown that the pattern $\hat{\mathbf{X}}^p$ is corrected at layer L. Now consider a pattern $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$.

$$\begin{aligned} n_{L_j}^q &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^q + \sum_{i=1}^{M} W_{L_j,L-1_i} O_{L-1_i}^q \\ &= C_j^p(\kappa + \epsilon_p - \sum_{i=1}^{N}(X_i^p)^2) + 2C_j^p \sum_{i=1}^{N}(X_i^p)(X_i^q) - C_j^p \sum_{i=1}^{N}(X_i^q)^2 + \kappa O_{L-1_j}^q \\ &= C_j^p(\kappa + \epsilon_p) + \kappa O_{L-1_j}^q - C_j^p \sum_{i=1}^{N}[(X_i^p)^2 - 2(X_i^p)(X_i^q) + (X_i^q)^2] \\ &= C_j^p(\kappa + \epsilon_p) + \kappa O_{L-1_j}^q - C_j^p [\sum_{i=1}^{N}(X_i^p - X_i^q)^2] \\ &= C_j^p(\kappa + \epsilon_p - \epsilon') + \kappa O_{L-1_j}^q \text{ where } \epsilon' = \sum_{i=1}^{N}(X_i^p - X_i^q)^2;\ \text{note } \epsilon' > \epsilon_p \\ &= \kappa' C_j^p + \kappa O_{L-1_j}^q \text{ where } \kappa + \epsilon_p - \epsilon' = \kappa' \qquad (3) \\ O_{L_j}^q &= sgn(n_{L_j}^q) \\ &= O_{L-1_j}^q \text{ since } \kappa' < \kappa \end{aligned}$$

Thus, for all patterns $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$, the outputs produced at layers L and $L-1$ are identical. We have shown the existence of a weight setting that is guaranteed to yield a reduction in the number of misclassified patterns whenever a new layer is added to the *Tower* network. We rely on the TLU weight training algorithm $\mathcal{A}$ to find such a weight setting. Since the training set is finite in size, eventual convergence to zero errors is guaranteed. □

WTA Output Strategy

We now show that even if the output of the *Tower* network is computed according to the WTA strategy, the weights for the output neurons in layer L given in equation (1) will ensure that the number of misclassifications is reduced by at least one.

Assume that the output vector $\mathbf{O}_{L-1}^p$ for the misclassified pattern $\hat{\mathbf{X}}^p$ is such that $O_{L-1_\beta}^p = 1$ and $O_{L-1_k}^p = -1,\ \forall k = 1 \ldots M, k \neq \beta$; whereas the target output $\mathbf{C}^p$ is such that $C_\gamma^p = 1$ and $C_l^p = -1,\ \forall l = 1 \ldots M, l \neq \gamma,$ and $\gamma \neq \beta$.

From equation (2) the net input for the neuron L_j is:

$$n^p_{L_j} = C^p_j(\kappa + \epsilon_p) + \kappa O^p_{L-1_j}$$

The net inputs for the output neurons L_γ, L_β, and L_j where $j = 1 \ldots M; j \neq \gamma, j \neq \beta$ are given by

$$\begin{aligned} n^p_{L_\gamma} &= C^p_\gamma(\kappa + \epsilon_p) + \kappa O^p_{L-1_\gamma} \\ &= \epsilon_p \\ n^p_{L_\beta} &= C^p_\beta(\kappa + \epsilon_p) + \kappa O^p_{L-1_\beta} \\ &= -\epsilon_p \\ n^p_{L_j} &= C^p_j(\kappa + \epsilon_p) + \kappa O^p_{L-1_j} \\ &= -2\kappa - \epsilon_p \end{aligned}$$

Since, the net input of neuron L_γ is higher than that of every other neuron in the output layer, we see that $O^p_{L_\gamma} = 1$ and $O^p_{L_j} = -1 \; \forall j \neq \gamma$. Thus pattern $\hat{\mathbf{X}}^p$ is correctly classified at layer L. Even if the output in response to pattern $\hat{\mathbf{X}}^p$ at layer $L-1$ had been $O^p_{L-1_j} = -1; \forall j = 1 \ldots M$, it is easy to see that given the weight setting for neurons in layer L, $\hat{\mathbf{X}}^p$ would be correctly classified at layer L.

Consider the pattern $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$ that is correctly classified at layer $L-1$ (i.e., $\mathbf{O}^q_{L-1} = \mathbf{C}^q$). From equation (3), the net input for neuron L_j is:

$$n^q_{L_j} = C^p_j(\kappa + \epsilon_p - \epsilon^{'}) + \kappa O^q_{L-1_j}$$

Since, $\kappa + \epsilon_p - \epsilon^{'} < \kappa$, it is easy to see that the neuron L_γ such that $O^q_{L-1_\gamma} = 1$ has the highest net input among all output neurons irrespective of the value assumed by C^p_γ. With this, $\mathbf{O}^q_L = \mathbf{O}^q_{L-1} = \mathbf{C}^q$. Thus, the classification of previously correctly classified patterns remains unchanged.

We have thus proved the convergence of the *Tower* algorithm even when the outputs are computed according to the WTA strategy.

3 Pyramid Algorithm

The 2-category *Pyramid* algorithm [Gallant, 1990] constructs a network in a manner similar to the *Tower* algorithm, except that each newly added neuron receives input from each of the N input neurons as well as the outputs of all the neurons in each of the preceding layers. The newly added neuron becomes the output of the network. As in the case of the *Tower* algorithm, the extension of the 2-category *Pyramid* algorithm to

handle M output categories and real-valued pattern attributes is quite straightforward. Each pattern is modified by appending the extra attribute (X^p_{N+1}). Each newly added layer of M neurons receives inputs from the $N+1$ input neurons and the outputs of each neuron in each of the previously added layers. The resulting *Pyramid* network is shown in Fig. 3.

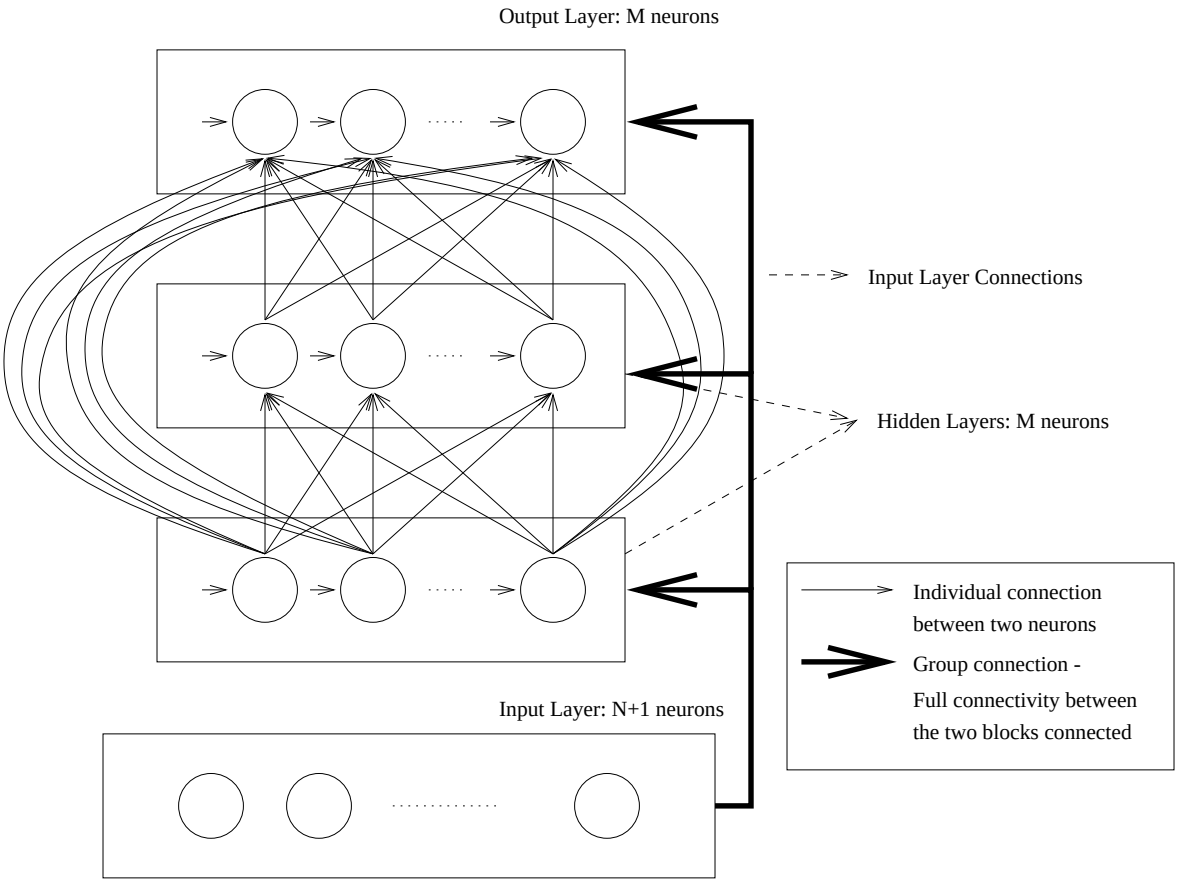

Figure 3: Pyramid Network

3.1 Multi-Category Pyramid Algorithm

1. Set the current output layer index $L = 0$.

2. Repeat the following steps until the desired training accuracy is achieved or the maximum number of hidden layers allowed is exceeded.

   (a) $L = L + 1$. Add M neurons to the network at layer L. This forms the new output layer of the network. Connect each neuron in the layer L to the $N+1$ input neurons and each neuron in each of the previous layers if they exist.

(b) Train the weights associated with the neurons in layer L (the rest of the weights in the network are left unchanged).

3.2 Convergence Proof

Theorem 2:
There exists a weight setting for neurons in the newly added layer L in the multi-category *Pyramid* network such that the number of patterns misclassified by the network with L layers is less than the number of patterns misclassified prior to the addition of the L^{th} layer (i.e., $\forall L > 1, \;\; e_L < e_{L-1}$).

Proof:
Define $\kappa = \max_{p,q} \sum_{i=1}^{N} (X_i^p - X_i^q)^2$. For each pattern $\hat{\mathbf{X}}^p$, define ϵ_p as $0 < \epsilon_p < \min_{p,q \neq p} \sum_{i=1}^{N} (X_i^p - X_i^q)^2$. It is clear that $0 < \epsilon_p < \kappa$ for all patterns $\hat{\mathbf{X}}^p$. Assume that a pattern $\hat{\mathbf{X}}^p$ was not correctly classified in layer $L-1$ (i.e., $\mathbf{C}^p \neq \mathbf{O}_{L-1}^p$). Consider the following weight setting for neuron L_j $(j = 1 \ldots M)$.

$$
\begin{aligned}
W_{L_j,0} &= C_j^p(\kappa + \epsilon_p - \sum_{i=1}^{N} (X_i^p)^2) \\
W_{L_j,I_i} &= 2C_j^p X_i^p \text{ for } i = 1 \ldots N \\
W_{L_j,I_{N+1}} &= -C_j^p \\
W_{L_j,L-i_k} &= 0 \text{ for } i = 2 \ldots L-1, \text{ and } k = 1 \ldots M \\
W_{L_j,L-1_j} &= \kappa \\
W_{L_j,L-1_k} &= 0 \text{ for } k = 1 \ldots M, k \neq j
\end{aligned}
\tag{4}
$$

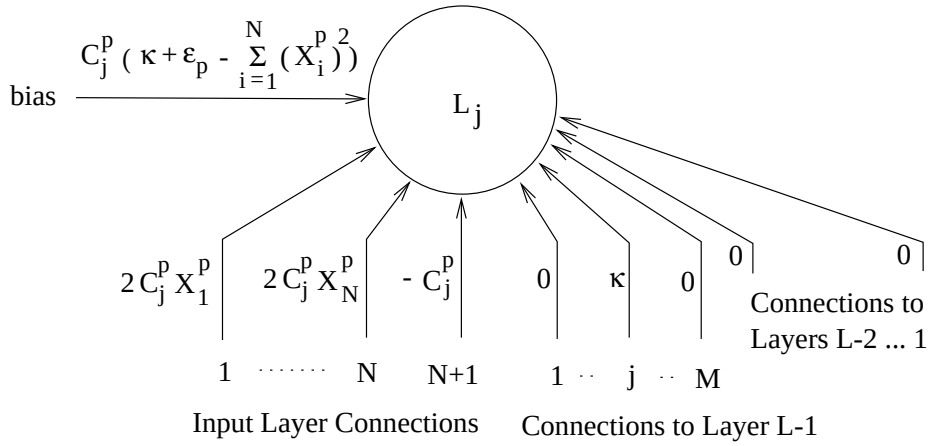

Figure 4: Weight setting for output neuron L_j in the Pyramid Network

This choice of weights for the output layer L reduces the multi-category *Pyramid* network to a multi-category *Tower* network. The convergence proof (for both the independent

and WTA output strategies) follows directly from the convergence proof of the *Tower* algorithm. □

4 Upstart Algorithm

The 2-category *Upstart* algorithm [Frean, 1990b] constructs a binary tree of threshold neurons. A simple extension of this idea to deal with M output categories would be to construct M independent binary trees (one for each output class). This approach fails to exploit the inter-relationships that might exist between the different outputs. We therefore follow an alternative approach [Frean, 1990b] using a single hidden layer instead of a binary tree. Since the original *Upstart* algorithm was presented for the case with binary valued patterns and binary TLUs, we will present our extension of this algorithm to M classes under the same binary valued framework[1]. Again, to handle patterns with real-valued attributes we consider the projection of the pattern vectors[2].

The extension of the *Upstart* algorithm to handle multiple output categories is described as follows[3]. First, an output layer of M neurons is trained using the algorithm $\mathcal{A}$. If all the patterns are correctly classified, the procedure terminates without the addition of any hidden neurons. If that is not the case, the output neuron (L_k) that makes the most number of errors (in the sense $C_k^p \neq O_{L_k}^p$) is identified. Depending on whether the neuron k is *wrongly-on* (i.e., $C_k^p = 0, O_{L_k}^p = 1$) or *wrongly-off* (i.e., $C_k^p = 1, O_{L_k}^p = 0$) more often, a wrongly-on corrector daughter (X) or a wrongly-off corrector daughter (Y) is added to the hidden layer and trained to correct some errors of neuron L_k. For each pattern $\hat{\mathbf{X}}^p$ in the training set, the target outputs (C_X^p and C_Y^p) for the X and Y daughters are determined as follows:

- If $C_k^p = 0$ and $O_{L_k}^p = 0$ then $C_X^p = 0$, $C_Y^p = 0$.
- If $C_k^p = 0$ and $O_{L_k}^p = 1$ then $C_X^p = 1$, $C_Y^p = 0$.
- If $C_k^p = 1$ and $O_{L_k}^p = 0$ then $C_X^p = 0$, $C_Y^p = 1$.
- If $C_k^p = 1$ and $O_{L_k}^p = 1$ then $C_X^p = 0$, $C_Y^p = 0$.

The daughter is trained using the algorithm $\mathcal{A}$. The daughter is connected to each neuron in the output layer and the output weights are retrained. The resulting network is shown in Fig. 5.

[1]The modification to handle bipolar valued patterns is straightforward with the only change being that instead of adding a X daughter or a Y daughter, a pair of X and Y daughters must be added at each time.

[2]An extension of the *Upstart* algorithm to handle patterns with real valued attributes using stereographic projection was originally proposed by [Saffery & Thornton, 1991].

[3]An earlier version of this algorithm appeared in [Parekh *et al.*, 1997b].

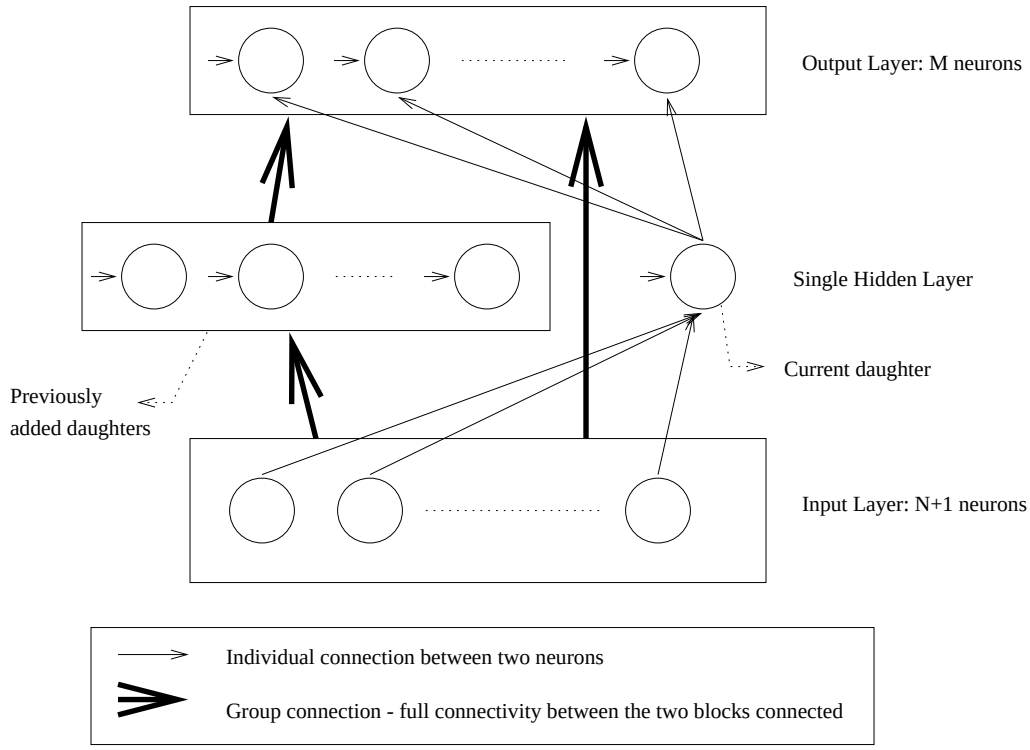

Figure 5: Upstart Network

4.1 Multi-Category Upstart Algorithm

1. Train a single layer network with M output neurons and $N+1$ input neurons.
2. If the desired training accuracy is not achieved thus far then repeat the following steps until the desired training accuracy is achieved or the maximum number of allowed neurons in the hidden layer is exceeded.
   (a) Determine the neuron L_k in the output layer that makes the most errors.
   (b) Add a X or a Y daughter depending on whether the neuron L_k is wrongly-on or wrongly-off more often. The daughter neuron is connected to the $N+1$ input units.
   (c) Construct the training set for the daughter neuron as described above and train it. Freeze the weights of this newly added daughter.
   (d) Connect the daughter neuron to each of the output neurons and retrain the output weights.

4.2 Convergence Proof

Theorem 3:
There exists a weight setting for the X daughter neuron and the output neurons in the multi-category *Upstart* algorithm such that the number of patterns misclassified by the network after the addition of the X daughter and the retraining of the output weights is less than the number of patterns misclassified prior to that.

Proof:
Assume that at some time during the training there is at least one pattern that is not correctly classified at the output layer L of M neurons[4]. Thus far, the hidden layer comprises of U_{L-1} daughter neurons. Assume also that the output neuron L_k is wrongly-on (i.e., it produces an output of 1 when the desired output is in fact 0) for a training pattern $\hat{\mathbf{X}}^p$. Let $\lambda > \sum_{j=1}^{M} abs(W_{L_j,0} + \sum_{k=1}^{N+1} W_{L_j,I_k} X_k^p + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^p)$ (i.e., λ is greater than the sum of the absolute values of the net inputs of all the neurons in the output layer L on the pattern $\hat{\mathbf{X}}^p$). A X daughter neuron is added to the hidden layer and trained so as to correct the classification of $\hat{\mathbf{X}}^p$ at the output layer. The daughter neuron is trained to output 1 for pattern $\hat{\mathbf{X}}^p$, and to output 0 for all other patterns. Next the newly added daughter neuron is connected to all output neurons and the output weights are retrained.

Consider the following weight setting for the daughter neuron:

$$\begin{aligned} W_{X,0} &= -\sum_{k=1}^{N} (X_k^p)^2 \\ W_{X,I_i} &= 2X_i^p \text{ for } i = 1 \ldots N \\ W_{X,I_{N+1}} &= -1 \end{aligned} \tag{5}$$

For pattern $\hat{\mathbf{X}}^p$:

$$\begin{aligned} n_X^p &= W_{X,0} + \sum_{k=1}^{N+1} W_{X,I_k} X_k^p \\ &= -\sum_{k=1}^{N} (X_k^p)^2 + \sum_{k=1}^{N} (2X_k^p) X_k^p - \sum_{k=1}^{N} (X_k^p)^2 \\ &= 0 \\ O_X^p &= 1 \text{ by definition of the binary threshold function} \end{aligned}$$

[4]In the case of the multicategory *Upstart* algorithm where only two layers viz. the output layer and the hidden layer are constructed, the output layer index is $L = 2$ and the hidden layer index is $L - 1 = 1$.

For any other pattern $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$:

$$\begin{aligned} n_X^q &= W_{X,0} + \sum_{k=1}^{N+1} W_{X,I_k} X_k^q \\ &= -\sum_{k=1}^{N} (X_k^p)^2 + \sum_{k=1}^{N} 2X_k^p X_k^q - \sum_{k=1}^{N} (X_k^q)^2 \\ &= -\sum_{k=1}^{N} (X_k^p - X_k^q)^2 \\ &< 0 \\ O_X^q &= 0 \text{ by definition of the binary threshold function} \end{aligned}$$

Let $\lambda > \sum_{j=1}^{M} [(abs(W_{L_j,0}) + \sum_{k=1}^{N+1} abs(W_{L_j,I_k}) + \sum_{k=1}^{U_{L-1}} abs(W_{L_j,L-1_k})]$ (i.e., λ is greater than the sum of the absolute values of the individual weights of all the neurons in the output layer L). Consider the following weight setting for connections between each output layer neuron and the newly trained X daughter:

$$W_{L_j,X} = 2(C_j^p - O_{L_j}^p)\lambda$$

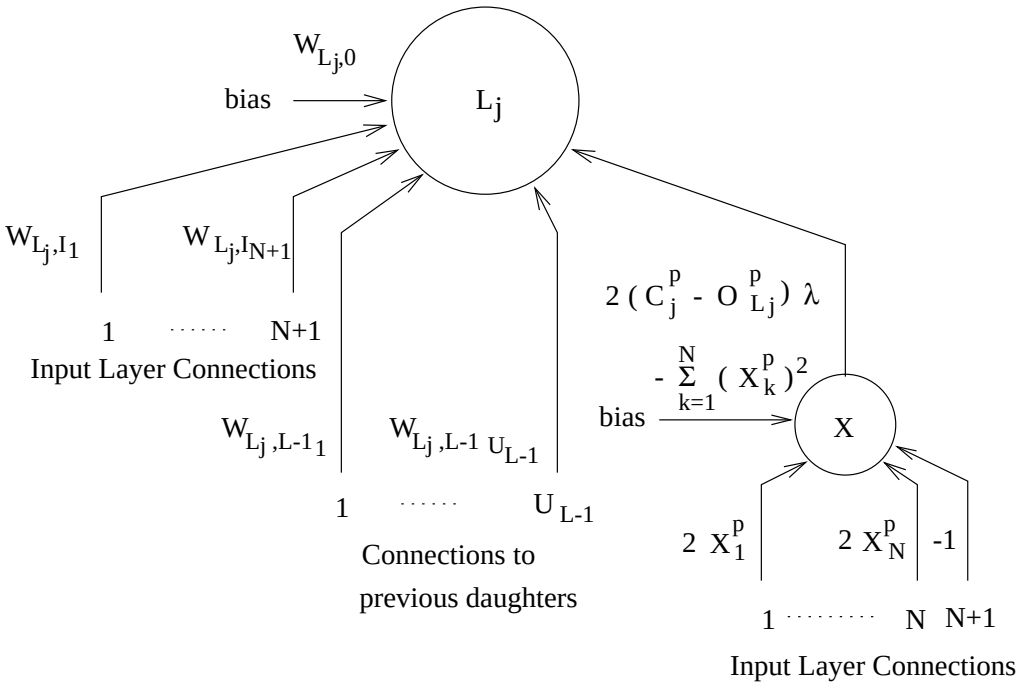

Figure 6: Weight setting for output neuron L_j in the Upstart Network

$O^p_{L_j}$ is the original output of neuron L_j in the output layer prior to adding the X daughter. Let us consider the new output of each neuron L_j in the output layer in response to pattern $\hat{\mathbf{X}}^p$

$$\begin{aligned} n^p_{L_j} &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^p_i + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O^p_{L-1_k} + 2(C^p_j - O^p_{L_j})\lambda O^p_X \\ &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^p_i + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O^p_{L-1_k} + 2(C^p_j - O^p_{L_j})\lambda(1) \end{aligned} \tag{6}$$

By the definition of λ we know that

$$-\lambda \leq \max_j [W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^p_i + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O^p_{L-1_k}] \leq \lambda \tag{7}$$

- If $C^p_j = O^p_{L_j}$ we see that the net input for neuron L_j remains the same as that before adding the daughter neuron and hence the output remains the same i.e., C^p_j.
- If $C^p_j = 0$ and $O^p_{L_j} = 1$, the net input for neuron L_j is $n^p_{L_j} \leq \lambda - 2\lambda$. Since $\lambda \geq 0$, the new output of L_j is 0 which is C^p_j.
- If $C^p_j = 1$ and $O^p_{L_j} = 0$, the net input for neuron j is $n^p_{L_j} \geq -\lambda + 2\lambda$. Since $\lambda \geq 0$, the new output of L_j is 1 which is C^p_j.

Thus pattern $\hat{\mathbf{X}}^p$ is corrected. Consider any other pattern $\hat{\mathbf{X}}^q$. We know that $O^q_X = 0$.

$$\begin{aligned} n^q_{L_j} &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^q_i + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O^q_{L-1_k} + 2(C^p_j - O^p_{L_j})\lambda O^q_X \\ &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^q_i + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O^q_{L-1_k} \end{aligned} \tag{8}$$

We see that the X daughter's contribution to the output neurons in the case of any patterns other than $\hat{\mathbf{X}}^p$ is zero. Thus the net input of each neuron in the output layer remains the same as it was before the addition of the daughter neuron and hence the outputs for patterns other than $\hat{\mathbf{X}}^p$ remain unchanged.

A similar proof can be presented for the case when a wrongly-off corrector (i.e., a Y daughter) is added to the hidden layer. Thus, we see that the addition of a daughter ensures that the number of misclassified patterns is reduced by at least one. Since the number of patterns in the training set is finite, the number of errors is guaranteed to eventually become zero. □

WTA Output Strategy

The mapping of the convergence proof for the *Upstart* algorithm to the case when the output neurons are trained using the WTA strategy is straightforward. In response to the pattern $\hat{\mathbf{X}}^p$, for which a wrongly-off corrector X is trained, the net input of neuron L_j is calculated as in equation (6). Given this and equation (7), it is easy to see that the neuron L_j for which $C_j^p = 1$ has the maximum net input among all output neurons and hence pattern $\hat{\mathbf{X}}^p$ is correctly classified.

For any other pattern $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$, the net input of all the output neurons is exactly the same as the net input prior to training the new X daughter (see equation (8)), the classification of pattern $\hat{\mathbf{X}}^q$ remains unchanged.

We have thus proved the convergence of the *Upstart* algorithm even when the outputs are computed according to the WTA strategy.

5 Perceptron Cascade Algorithm

The *Perceptron Cascade* algorithm [Burgess, 1994] draws on the ideas used in the *Upstart* algorithm and constructs a neural network that is topologically similar to the one built by the *Cascade correlation algorithm* [Fahlman & Lebiere, 1990]. However, unlike the *Cascade correlation algorithm* the *Perceptron Cascade* algorithm uses TLUs. Initially an output neuron is trained using the algorithm $\mathcal{A}$. If the output neuron does not correctly classify the training set, a daughter neuron (wrongly-on or wrongly-off as desired) is added and trained to correct some of the errors. The daughter neuron receives inputs from each of the input neurons and from each of the previously added daughters. As shown in Fig. 7 each daughter neuron is added to a new hidden layer during the construction of the *Perceptron Cascade* network. The targets for the daughter are determined exactly as in the case of the *Upstart* network.

The extension of the *Perceptron Cascade* algorithm to M output classes is relatively straight forward. First, an output layer of M neurons is trained. If all the patterns are correctly classified, the procedure terminates without the addition of any hidden neurons. If that is not the case, the output neuron, L_k, that makes the largest number of errors (in the sense that $C_k^p \neq O_{L_k}^p$) is identified and a daughter neuron (an X daughter if the neuron is wrongly-on more often or a Y daughter if the neuron is wrongly-off more often) is added to a new hidden layer and trained to correct some of the errors made by the output neurons. For each pattern $\hat{\mathbf{X}}^p$ in the training set, the target outputs for the daughter neuron are determined as in the *Upstart* algorithm. The daughter receives its inputs from each of the input neurons and from the outputs of each of the previously added daughters. After the daughter is trained it is connected to each of the M output neurons and the output weights are retrained. Fig. 7 shows the construction of a *Percep-*

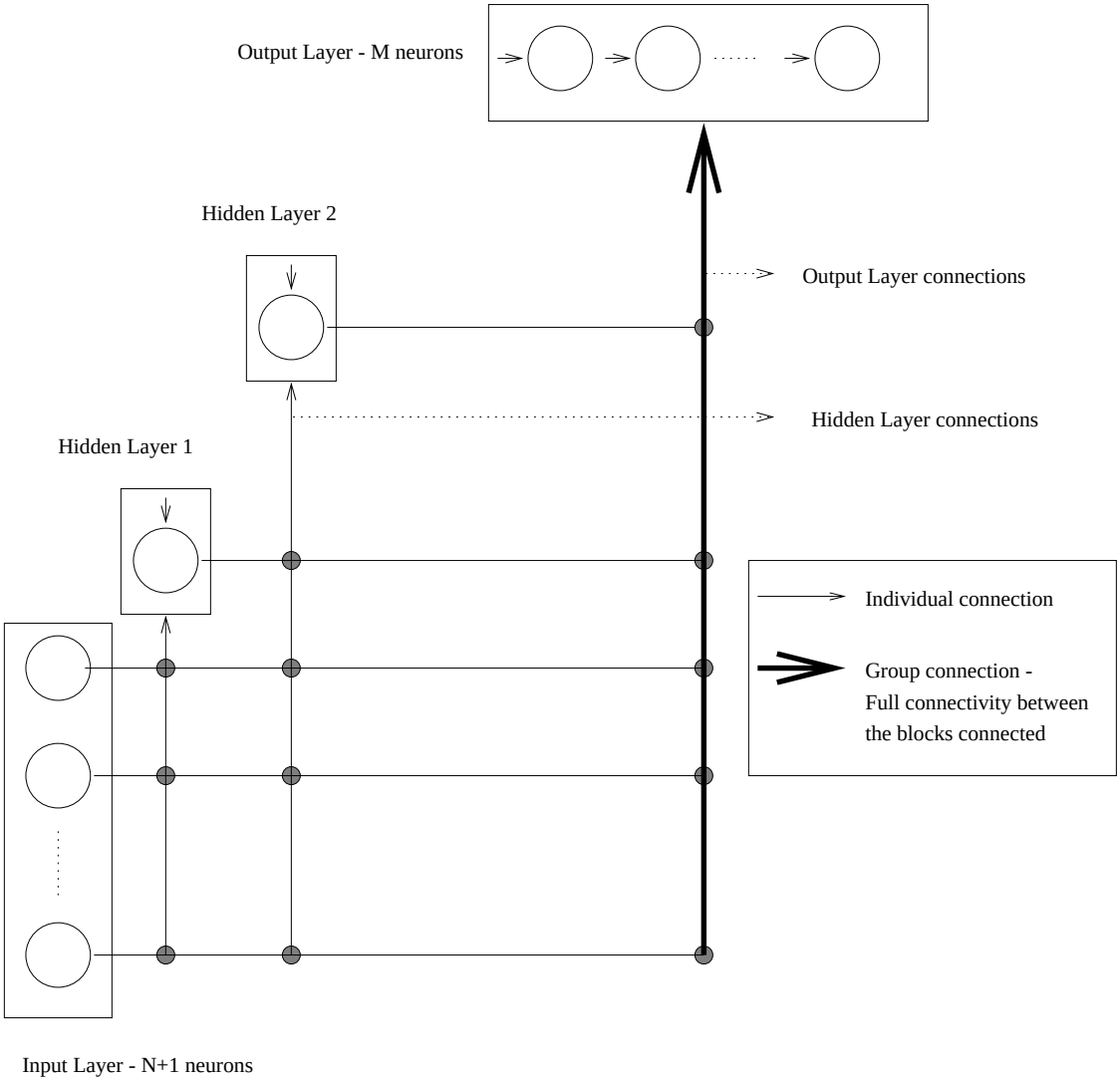

Figure 7: Perceptron Cascade Network

tron Cascade network. The extension to handle real-valued pattern attributes involves taking the projection of the patterns.

5.1 Multi-Category Perceptron Cascade Algorithm

1. Train a single layer network with M output neurons and $N + 1$ input neurons.
2. If the desired training accuracy is not achieved thus far then repeat the following steps until the desired training accuracy is achieved or the maximum number of hidden layers (each hidden layer comprises of a single daughter neuron) is exceeded.
   (a) Determine the output neuron L_k that makes the most errors.

(b) Add a X or a Y daughter in a new hidden layer immediately below the output layer depending on whether L_k is wrongly-on or wrongly-off more often. The daughter neuron is connected to all the $N+1$ input neurons and to all previously added daughter neurons.

(c) Construct the training set for the daughter neuron and train it. Freeze the weights of the daughter.

(d) Connect the daughter neuron to each of the output neurons and retrain the output weights.

5.2 Convergence Proof

Theorem 4:
There exists a weight setting for the X daughter neuron and the output neurons in the multi-category *Perceptron Cascade* algorithm such that the number of patterns misclassified by the network after the addition of the X daughter and the retraining of the output weights is less than the number of patterns misclassified prior that.

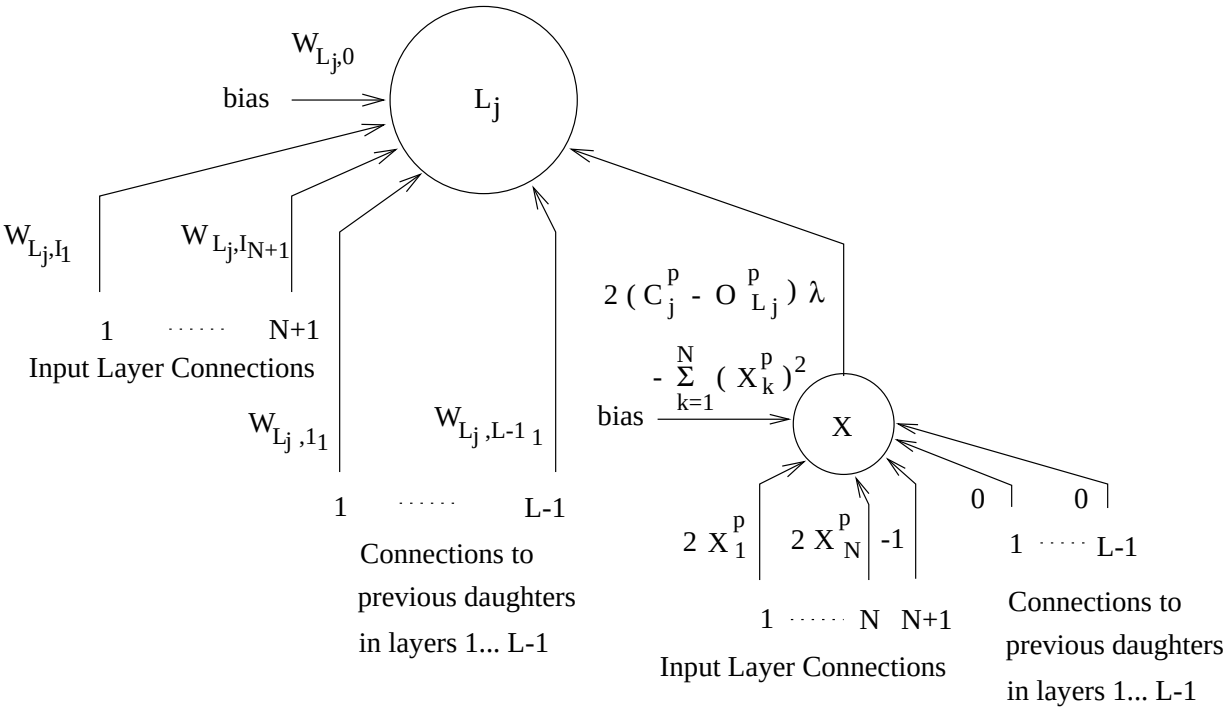

Figure 8: Weight setting for output neuron L_j in the Perceptron Cascade Network

Proof:
The *Perceptron Cascade* algorithm is similar to the *Upstart* algorithm except that each newly added daughter neuron is connected to all the previously added daughter neurons in addition to all the input neurons. If we set the weights connecting the daughter neuron

to all the previous daughter neurons to zero, the *Perceptron Cascade* algorithm would behave exactly as the *Upstart* algorithm. The convergence proof for the *Perceptron Cascade* algorithm (both in the case of the independent and WTA output strategies) thus follows directly from the proof of the *Upstart* algorithm. □

6 Tiling Algorithm

The *Tiling* algorithm [Mézard & Nadal, 1989] constructs a strictly layered network of threshold neurons. The bottom-most layer receives inputs from each of the N input neurons. The neurons in each subsequent layer receive inputs from those in the layer immediately below itself. Each layer maintains a *master neuron*. The network construction procedure ensures that the master neuron in a given layer correctly classifies more patterns than the master neuron of the previous layer. Ancillary neurons may be added to layers and trained to ensure a *faithful representation* of the training set. The *faithfulness* criterion simply ensures that no two training examples belonging to different classes produce identical output at any given layer. Faithfulness is clearly a necessary condition for convergence in strictly layered networks [Mézard & Nadal, 1989].

The proposed extension to multiple output classes involves constructing layers with M master neurons (one for each of the output classes)[5]. Unlike the other algorithms seen before, it is not necessary to take the projection of the input patterns to guarantee convergence for patterns with real-valued attributes. Sets of one or more ancillary neurons are trained at a time in an attempt to make the current layer faithful. Fig. 9 shows the construction of a *Tiling* network.

6.1 Multi-Category Tiling Algorithm

1. Train a layer of M master neurons each connected to the N input neurons.
2. If the master neurons can achieve the desired classification accuracy then stop.
3. Otherwise, if the current layer is not faithful, add ancillary neurons to the current layer to make it faithful as follows, else go to step 4.
   (a) Among all the unfaithful output vectors at the current output layer, identify the one that the largest number of input patterns map to. (An output vector is said to be unfaithful if it is generated by input patterns belonging to different classes).
   (b) Determine the set of patterns that generate the output vector identified in step 3(a) above. This set of patterns will form the training set for ancillary neurons.

[5] An earlier version of this algorithm appeared in [Yang *et al.*, 1996].

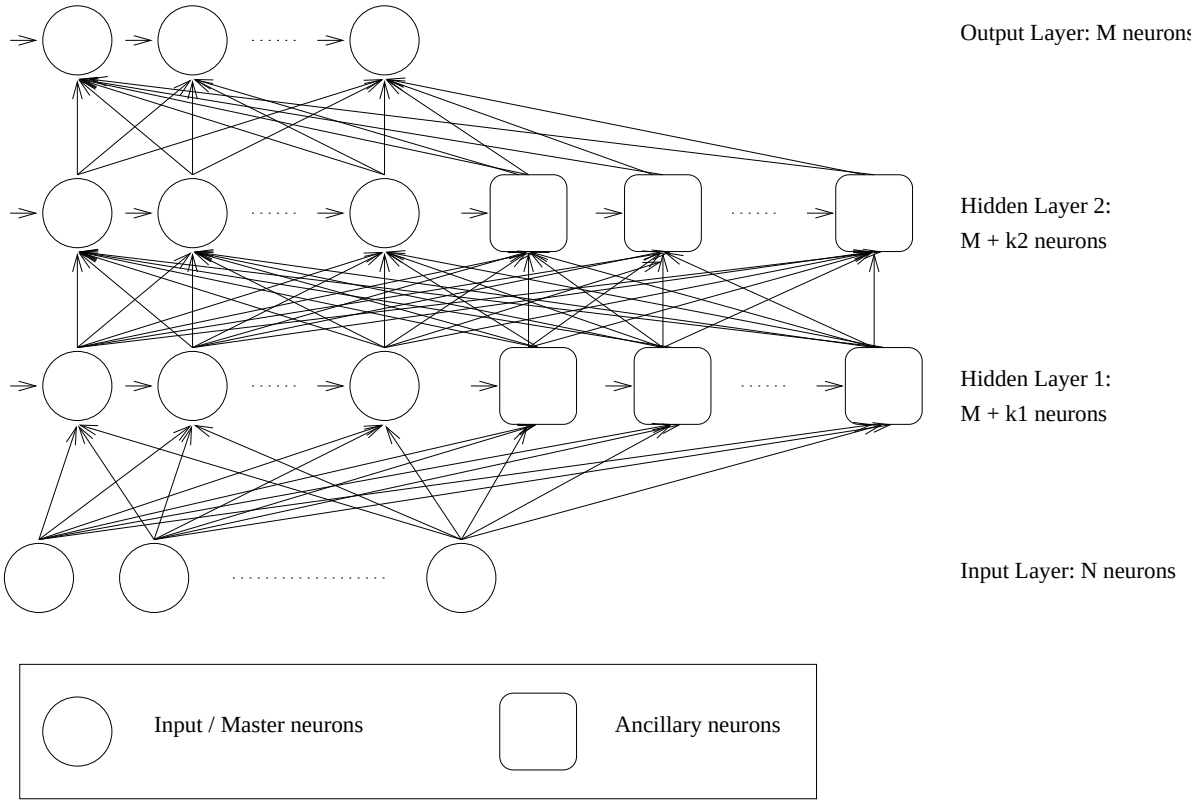

Figure 9: Tiling Network

(c) Add a set of k ($1 \leq k \leq M$) ancillary neurons where k is the number of target classes represented in the set of patterns identified above and train them.

(d) Repeat these last three steps (of adding and training ancillary neurons) till the output layer representation of the patterns is faithful.

4. Train a new layer of M master neurons that are connected to each neuron in the previous layer and go to step 2.

6.2 Convergence Proof

The convergence of the multi-category *Tiling* algorithm with real-valued pattern attributes to zero classification errors is proved in two parts: first we show that it is possible to obtain a faithful representation of the input pattern set (with real-valued attributes) at the first hidden layer. Then we depict that with each additional layer the number of classification errors is reduced by at least one.

In the *Tiling* algorithm each hidden layer contains M master neurons plus several ancillary neurons to achieve a faithful representation of the patterns in the layer. Let $\mathbf{\Pi}^p =< \pi_1^p, \pi_2^p, \ldots, \pi_{M+K}^p >$ (also called a prototype) be the representation of a subset

of patterns that have the same output in a layer (say A) with $U_A = M$ (master) + K (ancillary) neurons. $\pi_i^p = \pm 1$ for all $i = 1 \ldots (M + K)$.

Theorem 5a:
For any finite, non-contradictory dataset it is possible to train a layer of threshold neurons such that the outputs of these neurons provide a faithful representation of the pattern set.

Proof:
Assume that the training patterns represent a set S of N-dimensional pattern vectors. Further assume that training the M master neurons is unsuccessful in correctly classifying all the patterns and that all patterns are assigned to the same output class thereby causing the representation of the set S to be unfaithful. We now show that it is possible to add ancillary neurons (with appropriately set weights) that would result in a faithful representation of S for this layer of threshold neurons. Let $\mathbf{W} = \{W_0, W_1, \ldots, W_n\}$ designate the weight vector for a single TLU, T.

If there exists a pattern $\mathbf{X}^p$ belonging to the *convex hull* [6] of the set S such that for some attribute i $(i = 1, \ldots, n)$ $|X_i^p| > |X_i^q|$ for all $\mathbf{X}^q \in S$ and $\mathbf{X}^q \neq \mathbf{X}^p$ then $\mathbf{W} = \{-(X_i^p)^2, 0, \ldots, 0, X_i^p, 0, \ldots 0\}$ (i.e., all weights except W_0 and W_i set to 0) will cause T to output 1 for $\mathbf{X}^p$ and -1 for all other patterns.

If however, the set S is such that the highest value for each attribute is shared by at least two patterns then the above method for excluding a single pattern will not work. In this case, there must exist a pattern $\mathbf{X}^p$ in the convex hull of S that dominates all others in the sense that for each attribute i $X_i^p \geq X_i^q$ for all $\mathbf{X}^q$ in S. Clearly, $\mathbf{X}^p \cdot \mathbf{X}^p > \mathbf{X}^p \cdot \mathbf{X}^q$. The weights for T can be set to $\mathbf{W} = \{-\sum_{l=1}^{n}(X_l^p)^2, X_1^p, \ldots X_n^p\}$. With this weight setting T will output 1 for $\mathbf{X}^p$ and -1 for all other patterns.

Thus, pattern $\mathbf{X}^p$ is made to achieve a faithful representation. Similarly, another TLU can be trained to achieve a faithful representation for the pattern $\mathbf{X}^q \in S - \{\mathbf{X}^p\}$. It is clear that with the addition of $K = |S|$ hidden neurons a faithful representation of S will be achieved at the first hidden layer. □

Of course, in practice, by training a groups of one or more ancillary neurons using the algorithm $\mathcal{A}$ it is possible to attain a faithful representation of the input pattern set at the first hidden layer using far fewer TLUs as compared to the number of training patterns.

Theorem 5b:

[6] The convex hull for a set of points Q is the smallest convex polygon P such that each point in Q lies either on the boundary of P or in its interior.

There exists a weight setting for the master neurons in the newly added layer L in the multi-category *Tiling* network such that the number of patterns misclassified by the network with L layers is less than the number of patterns misclassified prior to the addition of the L^{th} layer (i.e., $\forall L > 1,\ e_L < e_{L-1}$).

Proof:
Consider a prototype $\mathbf{\Pi}^p$ for which the master neurons in layer $L-1$ do not yield the correct output. i.e., $< \pi_1^p, \pi_2^p, \ldots, \pi_M^p > \neq < C_1^p, C_2^p, \ldots, C_M^p >$. The following weight setting for the master neuron L_j $(j = 1 \ldots M)$ results in correct output for prototype $\mathbf{\Pi}^p$. Also, this weight setting ensures that the outputs of all other prototypes, $\mathbf{\Pi}^q$, for which the master neurons at layer $L-1$ produce correct outputs (i.e., $< \pi_1^q, \pi_2^q, \ldots, \pi_M^q > = < C_1^q, C_2^q, \ldots, C_M^q >$), are unchanged.

$$\begin{aligned} W_{L_j,0} &= 2C_j^p \\ W_{L_j,L-1_k} &= C_j^p \pi_k^p \text{ for } k = 1 \ldots U_{L-1}, k \neq j \\ W_{L_j,L-1_j} &= U_{L-1} \end{aligned} \tag{9}$$

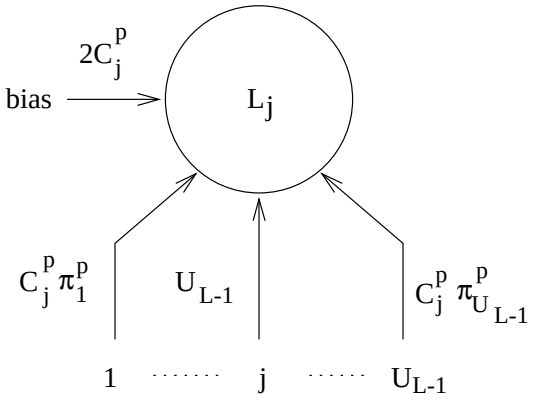

Figure 10: Weight setting for output neuron L_j in the Tiling Network

For prototype $\mathbf{\Pi}^p$:

$$\begin{aligned} n_{L_j}^p &= W_{L_j,0} + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} \pi_k^p \\ &= 2C_j^p + U_{L-1}\pi_j^p + (U_{L-1} - 1)C_j^p \\ &= U_{L-1}\pi_j^p + (U_{L-1} + 1)C_j^p \\ O_{L_j}^p &= sgn(n_{L_j}^p) \\ &= C_j^p \end{aligned} \tag{10}$$

For prototype $\mathbf{\Pi}^q$ (as described above) $\neq \mathbf{\Pi}^p$:

$$\begin{aligned} n^q_{L_j} &= W_{L_j,0} + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k}\pi^q_k \\ &= 2C^p_j + U_{L-1}\pi^q_j + \sum_{k=1,k\neq j}^{U_{L-1}} W_{L_j,L-1_k}\pi^q_k \\ &= 2C^p_j + U_{L-1}\pi^q_j + \sum_{k=1,k\neq j}^{U_{L-1}} C^p_j\pi^p_k\pi^q_k \end{aligned} \tag{11}$$

CASE I:
$\pi^q_j \neq \pi^p_j$ and $\pi^q_k = \pi^p_k$ for $1 \leq k \leq U_{L-1}, k \neq j$.
For example,

$$\begin{aligned} \mathbf{\Pi}^p &= < \underbrace{-1, \overbrace{+1}^{l}, -1, \ldots, \overbrace{+1}^{j}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} > \\ \mathbf{\Pi}^q &= < \underbrace{-1, \overbrace{+1}^{l}, -1, \ldots, \overbrace{-1}^{j}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} > \end{aligned}$$

Since π^q is correctly classified at layer $L-1$ whereas π^p is not, $\pi^q_j = C^p_j$ (this follows from the fact that $\pi^q_j = -\pi^p_j$ and $C^p_j = -\pi^p_j$).

$$\begin{aligned} n^q_{L_j} &= 2C^p_j + U_{L-1}\pi^q_j + \sum_{k=1,k\neq j}^{U_{L-1}} C^p_j\pi^p_k\pi^q_k \\ &= 2C^p_j + U_{L-1}\pi^q_j + (U_{L-1}-1)C^p_j \\ &= U_{L-1}\pi^q_j + (U_{L-1}+1)C^p_j \\ &= (2U_{L-1}+1)\pi^q_j \text{ since } \pi^q_j = C^p_j \\ O^q_{L_j} &= sgn(n^q_{L_j}) \\ &= \pi^q_j \end{aligned}$$

CASE II:
$\pi^q_l \neq \pi^p_l$ for some l, $1 \leq l \leq U_{L-1}, l \neq j$ and $\pi^q_k = \pi^p_k$ for all k, $1 \leq k \leq U_{L-1}, k \neq j, k \neq l$
For example,

$$\begin{aligned} \mathbf{\Pi}^p &= < \underbrace{-1, -1, -1, \ldots, \overbrace{+1}^{j}, \ldots, -1}_{M}, \underbrace{-1, \ldots, \overbrace{-1}^{l}, \ldots, +1}_{K} > \\ \mathbf{\Pi}^q &= < \underbrace{-1, -1, -1, \ldots, \overbrace{+1}^{j}, \ldots, -1}_{M}, \underbrace{-1, \ldots, \overbrace{+1}^{l}, \ldots, +1}_{K} > \end{aligned}$$

In this case, $\sum_{k=1,k\neq j}^{U_{L-1}} C_j^p \pi_k^p \pi_k^q \leq (U_{L-1}-3)C_j^p$

$$\begin{aligned} n_{L_j}^q &= 2C_j^p + U_{L-1}\pi_j^q + \sum_{k=1,k\neq j}^{U_{L-1}} C_j^p \pi_k^p \pi_k^q \\ &\leq 2C_j^p + U_{L-1}\pi_j^q + (U_{L-1}-3)C_j^p \\ &\leq (U_{L-1}-1)C_j^p + U_{L-1}\pi_j^q \\ O_{L_j}^q &= sgn(n_{L_j}^q) \\ &= \pi_j^q \text{ since } U_{L-1}\pi_j^q \text{ dominates } (U_{L-1}-1)C_j^p \end{aligned}$$

Once again we rely on the algorithm $\mathcal{A}$ to find an appropriate weight setting. With the above weights the previously incorrectly classified prototype, $\mathbf{\Pi}^p$, would be corrected and all other prototypes that were correctly classified would be unaffected. This reduces the number of incorrect prototypes by at least one (i.e., $e_L < e_{L-1}$). Since the training set is finite, the number of prototypes must be finite, and with a sufficient number of layers the *Tiling* algorithm would eventually converge to zero classification errors. □

WTA Output Strategy

For the incorrectly classified prototype $\mathbf{\Pi}^p$ described earlier assume that $\pi_\beta^p = 1, 1 \leq \beta \leq M$ and $\pi_j^p = -1\ \forall j = 1 \ldots M, j \neq \beta$. Clearly, $C_\beta^p = -1$ and $\exists\gamma\ 1 \leq \gamma \leq M, \gamma \neq \beta$ such that $C_\gamma^p = 1$. Given the weight settings for the master neurons in layer L in equation (9), the net input of neuron L_j in response to the prototype $\mathbf{\Pi}^p$ as given in equation (10) is

$$\begin{aligned} n_{L_j}^p &= U_{L-1}\pi_j^p + (U_{L-1}+1)C_j^p \\ n_{L_\gamma}^p &= U_{L-1}(-1) + (U_{L-1}+1)(1) \\ &= 1 \\ n_{L_\beta}^p &= U_{L-1}(1) + (U_{L-1}+1)(-1) \\ &= -1 \text{ where } 1 \leq \beta \leq M, \beta \neq \gamma \\ n_{L_k}^p &= U_{L-1}(-1) + (U_{L-1}+1)(-1) \text{ for } k = 1 \ldots M, k \neq \gamma, k \neq \beta \\ &= -2U_{L-1} - 1 \end{aligned}$$

The master neuron L_γ has the highest net input among all master neurons in layer L which means that $O_{L_\gamma}^p = 1$ and $O_{L_j}^p = -1, \forall j = 1 \ldots M, j \neq \gamma$ and $\mathbf{C}^p = \mathbf{O}_L^p$. Thus, the prototype $\mathbf{\Pi}^p$ is now correctly classified.

For the prototype $\mathbf{\Pi}^q$ that is correctly classified at layer $L-1$ (as described earlier), since $\mathbf{\Pi}^q \neq \mathbf{\Pi}^p$, it is clear that $\pi_\beta^q = -1$ and $\exists\alpha\ 1 \leq \alpha \leq M, \alpha \neq \beta$ such that $\pi_\alpha^q = 1$. The net input of the master neurons at layer L in response to the prototype $\mathbf{\Pi}^q$ as

calculated in equation (11) is

$$n_{L_j}^q = 2C_j^p + U_{L-1}\pi_j^q + \sum_{k=1,k\neq j}^{U_{L-1}} C_j^p \pi_k^p \pi_k^q$$

CASE I: Assume that $\alpha = \gamma$ (where $C_\gamma^p = 1$). For example,

$$\mathbf{\Pi}^p = < \underbrace{-1, \overbrace{-1}^{\alpha=\gamma}, -1, \ldots, \overbrace{+1}^{\beta}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} >$$

$$\mathbf{\Pi}^q = < \underbrace{-1, \overbrace{+1}^{\alpha=\gamma}, -1, \ldots, \overbrace{-1}^{\beta}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} >$$

In this case $(2M - U_{L-1} - 3) \leq [\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \leq (U_{L-1} - 3)$. The net input for the output neuron L_α is

$$\begin{aligned} n_{L_\alpha}^q &= 2C_\alpha^p + U_{L-1}\pi_\alpha^q + C_\alpha^p [\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &= 2(1) + U_{L-1}(1) + (1)[\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &\geq 2 + U_{L-1} + 2M - U_{L-1} - 3 \\ &\geq 2M - 1 \end{aligned}$$

Similarly, the net input for any neuron j (other than α) in the output layer is given by

$$\begin{aligned} n_{L_j}^q &= 2C_j^p + U_{L-1}\pi_j^q + C_j^p [\sum_{k=1,k\neq j}^{U_{L-1}} \pi_k^p \pi_k^q] \text{ for } j = 1 \ldots M, j \neq \alpha \\ &= 2(-1) + U_{L-1}(-1) + (-1)[\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &\leq -2 - U_{L-1} + (-1)(2M - U_{L-1} - 3) \\ &\leq 1 - 2M \end{aligned}$$

Since $M \geq 3$ we see that the net input of neuron L_α is higher than the net input of any other master neuron in the output layer. Thus, $O_{L_\alpha}^q = 1$ and $O_{L_j}^q = -1 \; \forall j = 1 \ldots M, j \neq \alpha$ which means that $\mathbf{C}^q = \mathbf{O}_L^q$ as desired.

CASE II: Assume $\alpha \neq \gamma$ (where $C_\gamma^p = 1$). For example,

$$\mathbf{\Pi}^p = < \underbrace{-1, \overbrace{-1}^{\gamma}, -1, \ldots, \overbrace{+1}^{\beta}, \ldots, \overbrace{-1}^{\alpha}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} >$$

$$\mathbf{\Pi}^q \;=\; < \underbrace{-1, \overbrace{-1}^{\gamma}, -1, \ldots, \overbrace{-1}^{\beta}, \ldots, \overbrace{+1}^{\alpha}, \ldots, -1}_{M}, \underbrace{-1, \ldots, +1}_{K} >$$

In this case $C_\alpha^q = 1$; $(2M - U_{L-1} - 3) \leq [\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \leq (U_{L-1} - 3)$ and $(2M - U_{L-1} - 5) \leq [\sum_{k=1,k\neq\gamma}^{U_{L-1}} \pi_k^p \pi_k^q] \leq (U_{L-1} - 5)$.

The net input for output neuron L_α is

$$\begin{aligned} n_{L_\alpha}^q &= 2C_\alpha^p + U_{L-1}\pi_\alpha^q + C_\alpha^p[\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &= 2(-1) + U_{L-1}(1) + (-1)[\sum_{k=1,k\neq\alpha}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &\geq -2 + U_{L-1} - 1(U_{l-1} - 3) \\ &\geq 1 \end{aligned}$$

Analogously, the net input for the output neuron L_γ is

$$\begin{aligned} n_{L_\gamma}^q &= 2C_\gamma^p + U_{L-1}\pi_\gamma^q + C_\gamma^p[\sum_{k=1,k\neq\gamma}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &= 2(1) + U_{L-1}(-1) + 1[\sum_{k=1,k\neq\gamma}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &\leq 2 - U_{L-1} + (U_{l-1} - 5) \\ &\leq -3 \end{aligned}$$

Finally, the net input of output neuron L_j where $j \neq \alpha, j \neq \gamma$ is given by

$$\begin{aligned} n_{L_j}^q &= 2C_j^p + U_{L-1}\pi_j^q + C_j^p[\sum_{k=1,k\neq j}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &= 2(-1) + U_{L-1}(-1) + (-1)[\sum_{k=1,k\neq j}^{U_{L-1}} \pi_k^p \pi_k^q] \\ &\leq -2 - U_{L-1} - (2M - U_{L-1} - 3) \\ &\leq -2M + 1 \end{aligned}$$

Again, since $M \geq 3$ we see that the net input of neuron L_α is higher than the net input of any other neuron in the output layer. Thus, $O_{L_\alpha}^q = 1$ and $O_{L_j}^q = -1 \; \forall j = 1 \ldots M, j \neq \alpha$ which means that $\mathbf{C}^q = \mathbf{O}_L^q$ as desired. Thus we have shown that even if the output of the master neurons is computed according the the WTA strategy there is a weight setting for a newly added group of master neurons which will reduce the number of misclassifications by at least one.

7 Sequential Learning Algorithm

The *Sequential* learning algorithm [Marchand *et al.*, 1990] offers an alternative scheme for network construction where instead of training neurons to correctly classify a maximal subset of the training patterns, the idea is to train neurons to sequentially exclude patterns belonging to one class from the remaining patterns. The algorithm constructs a two layer network of threshold neurons where the hidden layer neurons are trained to sequentially exclude patterns belonging to a one class. When all the patterns in the training set have been thus excluded, the internal representation of the patterns at the hidden layer is guaranteed to be linearly separable. The weights of the single output layer neuron are fixed to correctly classify all patterns. Recently, Poulard has shown that a variation of the *Barycentric correction procedure* can be used effectively in *Sequential* learning to exclude as many patterns belonging to one class as possible [Poulard, 1995].

The extension of the *Sequential* learning algorithm to multiple output categories follows the same principles as the original version. Using a simple modification of the *Barycentric correction procedure*, hidden neurons can be trained to exclude patterns belonging to one of the M classes from the remaining patterns. Once all the patterns in the training set have been excluded by the hidden layer neurons, the output layer with M neurons can be constructed to correctly classify all patterns. As in the case of the *Tiling* algorithm, it is not necessary to consider the projection of the training patterns to prove the convergence for patterns with real-valued attributes. Fig. 11 depicts a network constructed by *Sequential* learning.

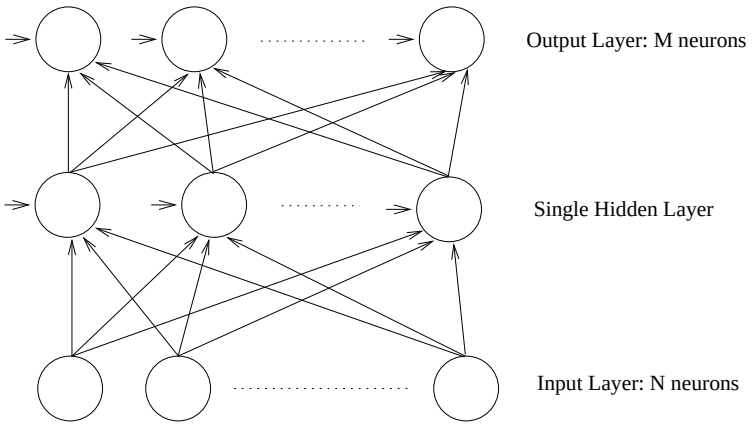

Figure 11: Sequential Network

7.1 Multi-Category Sequential Learning Algorithm

1. $i \leftarrow 0$.

2. Initialize S to the entire set of training patterns.

3. While $S \neq \phi$ do

   (a) Train a pool of M neurons using the *Barycentric correction procedure* (*Sequential* learning version). Neuron k ($k = 1, \ldots, M$) is trained to exclude as many patterns belonging to Ψ_k from the remaining patterns in S as possible.

   (b) Pick the neuron (trained in the previous step) that excludes the largest subset of patterns in S and designate it as neuron i in the hidden layer. Let E^i be the set of patterns excluded by the hidden layer neuron i.

   (c) $S \leftarrow S - E^i$.

   (d) $i = i + 1$

4. Construct the output layer with M neurons with each neuron connected to every neuron in the hidden layer.

5. Set the weights for the output layer neurons as follows

$$\begin{aligned} W_{L_j,L-1_k} &= \begin{cases} 2^{U_{L-1}+1-k} & \text{if neuron } L-1_k \text{ excludes } \Psi_j \\ -2^{U_{L-1}+1-k} & \text{otherwise} \end{cases} \\ W_{L_j,0} &= \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} \end{aligned} \tag{12}$$

7.2 Convergence Proof

We prove the convergence of this algorithm in two parts. Firstly, we show that it is possible to construct a hidden layer to sequentially exclude all patterns in the training set. Then we show that if the output layer is constructed as described in the algorithm, then all the patterns in the training set are correctly classified.

7.2.1 Construction of the Hidden Layer

Given the training set S for the neuron i of the hidden layer, intuitively it is clear that a weight setting exists for which one pattern belonging to the *convex hull* of the set S can be excluded from the rest. The proof of theorem 5a can be used directly to show that it is possible to construct a layer of threshold neurons that sequentially excludes any finite, non-contradictory pattern set.

7.2.2 Construction of the Output Layer

Consider that the hidden layer $L-1$ with U_{L-1} neurons is trained to sequentially exclude all patterns. The output layer L with M neurons is constructed with each neuron connected to all the U_{L-1} neurons in the hidden layer. Given that the weights of the output layer neurons are set as described in the algorithm we now show that all patterns belonging to the training set are correctly classified by the network.

Theorem 6: (*Sequential Learning Theorem*)
The internal representation of the training patterns that are excluded sequentially by the neurons in the single hidden layer are *linearly separable*.

Proof:
Let $\mathbf{X}^p$ be a pattern belonging to Ψ_j ($j = 1, \ldots, M$) and excluded by neuron $L-1_k$ ($k = 1, \ldots, U_{L-1}$). Clearly, for pattern $\mathbf{X}^p$, the hidden neurons $L-1_1, L-1_2, \ldots L-1_{k-1}$ output -1, the neuron $L-1_k$ outputs 1, and the hidden neurons $L-1_{k+1}, \ldots, L_{U_{L-1}}$ output 1 or -1. The net input of the output neuron L_j is:

$$\begin{aligned}
n^p_{L_j} &= W_{L_j,0} + \sum_{l=1}^{U_{L-1}} W_{L_j,L-1_l} O^p_{L-1_l} \\
&= \sum_{l=1}^{U_{L-1}} W_{L_j,L-1_l} + \sum_{l=1}^{k-1} W_{L_j,L-1_l}(-1) + W_{L_j,L-1_k} + \sum_{l=k+1}^{U_{L-1}} W_{L_j,L-1_l} O^p_{L-1_l} \\
&= \sum_{l=k+1}^{U_{L-1}} W_{L_j,L-1_l} + 2W_{L_j,L-1_k} + \sum_{l=k+1}^{U_{L-1}} W_{L_j,L-1_l} O^p_{L-1_l} \\
&\geq 2W_{L_j,L-1_k} \\
&> 0
\end{aligned} \tag{13}$$

Thus, the output neuron L_j's response to pattern $\mathbf{X}^p$ is 1. The net input of any other output neuron L_i ($i = 1, \ldots, M$ and $i \neq j$) is

$$\begin{aligned}
n^p_{L_i} &= W_{L_i,0} + \sum_{l=1}^{U_{L-1}} W_{L_i,L-1_l} O^p_{L-1_l} \\
&= \sum_{l=1}^{U_{L-1}} W_{L_i,L-1_l} + \sum_{l=1}^{k-1} W_{L_i,L-1_l}(-1) + W_{L_i,L-1_k} + \sum_{l=k+1}^{U_{L-1}} W_{L_i,L-1_l} O^p_{L-1_l} \\
&= \sum_{l=k+1}^{U_{L-1}} W_{L_i,L-1_l} + 2W_{L_i,L-1_l} + \sum_{l=k+1}^{U_{L-1}} W_{L_i,L-1_k} O^p_{L-1_l} \\
&= \sum_{l=k+1}^{U_{L-1}} W_{L_i,L-1_l} - 2|W_{L_i,L-1_l}| + \sum_{l=k+1}^{U_{L-1}} W_{L_i,L-1_k} O^p_{L-1_l} \text{ since } (W_{L_i,L-1_k} < 0) \\
&\leq 2 \sum_{l=k+1}^{U_{L-1}} |W_{L_i,L-1_l}| - 2|W_{L_i,L-1_k}| \\
&< 0 \text{ since } |W_{L_i,L-1_k}| > \sum_{l=k+1}^{U_{L-1}} |W_{L_i,L-1_l}|
\end{aligned} \tag{14}$$

Thus, each output neuron other than L_j has an output of -1 in response to pattern $\mathbf{X}^p$ and the network correctly classifies $\mathbf{X}^p$ as belonging to Ψ_j. Since each pattern is thus

correctly classified we have demonstrated that the internal representation of the training patterns that are excluded sequentially by the neurons in the single hidden layer are linearly separable. □

WTA Output Strategy

In the case of the *Sequential* learning algorithm, the weight assignment for the output weights from equation (12) ensures that for a pattern $\mathbf{X}^p$ belonging to Ψ_j, the net input of output neuron (L_j) is greater than 0 (see equation (13)) and the net input of all other neurons ($L_k, k = 1 \ldots M, k \neq j$) is less than 0 (see equation (14)). Clearly, we see that pattern $\mathbf{X}^p$ is correctly classified even if the output is computed using the WTA strategy.

8 Constructive Algorithms in Practice

The preceding discussion has focused on provably convergent constructive learning algorithms to handle real-valued, multi-category pattern classification problems. The algorithms differ from one another chiefly in the criteria used to decide when and where to add a neuron to an existing network, and the method used to train individual neurons. A systematic experimental study of constructive algorithms aimed at a thorough characterization of their implicit inductive, representational, and search biases (that arise from the construction procedures employed by the different algorithms) is beyond the scope of this paper. Such a study would entail, among other things, a careful experimental analysis of each constructive algorithm for different choices of single neuron training algorithm (e.g., *Pocket algorithm* with *ratchet modification*, *Thermal perceptron algorithm*, *Barycentric correction procedure*, and perhaps other variants designed for synergy with specific network construction strategies) and different output representations (e.g., independent output neurons versus WTA). This is the subject of [Parekh *et al.*, 1997a]. In what follows, we explore some practical issues that arise in the application of constructive learning algorithms and present the results of a few experiments designed to address the following key issues.

1. The convergence proofs presented here rely on two factors: The ability of the network construction strategy to connect a new neuron to an existing network so as to guarantee the existence of weights that will enable the added neuron to improve the resulting network's classification accuracy and the TLU weight training algorithm's ability to find such a weight setting. Finding an optimal weight setting for each added neuron such that the classification error is maximally reduced when the the data is non-separable is an NP-hard problem [Siu *et al.*, 1995]. Thus, practical algorithms for training threshold neurons are heuristic in nature. This makes it important to study the convergence of the proposed constructive algorithms in practice. We trained constructive networks on non-linearly separable datasets that require highly nonlinear decision surfaces.

2. It is important to examine whether constructive algorithms yield in practice, networks that are significantly smaller than would be the case if a new neuron is recruited to memorize each pattern in the training set. Comparison of the size of the networks generated by the algorithms with the number of patterns in the training set would, at least partially, answer this question.

3. Regardless of the convergence of the constructive learning algorithms to zero classification errors, a question of practical interest is the algorithms' ability to improve generalization on the test set as the network grows in size. One would expect *over-fitting* to set in eventually as neurons continue to get added in an attempt to reduce the classification error, but we wish to examine whether the addition of neurons improves generalization before over-fitting sets in. Experiments were designed to examine the generalization behavior of constructive algorithms on non-linearly separable datasets.

Another important issue, especially in the case of large pattern sets, is that of training time. Since our experiments were not designed for optimal performance in terms of training time, it is difficult to make a definitive statement comparing the training speeds of the different algorithms. We have identified some important factors that address this issue. For a detailed description of the datasets and the training methodology please see the appendix (sections A.2 and A.3).

8.1 Convergence Properties

Tables 1, 2, and 3 summarize the performance of the constructive algorithms on the **r5**, **3-circles** and **ionosphere** datasets respectively. In each case, the networks were trained to attain 100% classification accuracy on the training set and the network size (number of neurons excluding the input neurons), training time (in seconds), and generalization accuracy (the fraction of the test set that was correctly classified by the network) were recorded. These tables demonstrate that the constructive algorithms are indeed capable of converging to zero classification errors while generating sufficiently compact networks.

Algorithm	Network Size	Time
Tower	11.03±0.73	10.90±0.95
Pyramid	10.58±0.88	10.63±1.21
Upstart	10.64±0.57	52.34±4.83
Cascade	9.78±0.4	49.33±2.68
Tiling	14.95±1.17	9.49±0.76
Sequential	10.92±0.62	65.23±6.98

Table 1: Average Network Size and Training Time for the **r5** dataset

Algorithm	Network Size	Training Time	Test Accuracy
Tower	6.00±0.00	134.96±1.52	99.79±0.09
Pyramid	6.00±0.00	137.43±3.36	99.74±0.23
Upstart	19.00±15.53	1227.56±774.52	99.03±0.76
Cascade	8.38±5.52	690.28±533.57	99.14±1.15
Tiling	45.60±7.76	561.44±71.32	95.37±0.92
Sequential	44.68±6.26	1550.90±1282.35	94.44±1.13

Table 2: Performance of the Constructive Algorithms on the **3-circles** dataset

Algorithm	Network Size	Training Time	Test Accuracy
Tower	5.68±1.65	97.94±28.25	94.8±1.52
Pyramid	5.04±0.98	90.16±19.03	94.84±1.17
Upstart	3.04±0.45	133.77±28.39	94.12±1.89
Cascade	3.28±0.61	148.56±39.17	93.03±2.10
Tiling	8.76±1.48	86.99±9.43	89.64±3.46
Sequential	5.08±0.4	106.17±29.39	91.62±2.53

Table 3: Performance of the Constructive Algorithms on the **ionosphere** dataset

Certain constructive algorithms experienced difficulty in successfully classifying the entire training set in the case of some datasets (e.g., **iris** and **segmentation**). In tables 4 and 5 we describe the results of those constructive algorithms that did manage to converge successfully to zero classification errors on these training sets.

Algorithm	Network Size	Training Time	Test Accuracy
Tiling	9.76±3.27	17.19±2.16	96.08±1.35
Sequential	7.0±0.0	80.8±26.24	90.32±0.75

Table 4: Performance of the Constructive Algorithms on the **iris** dataset

Owing to the inherent bias of the network construction strategy, there might be a particular network construction strategies that are favorably disposed towards certain datasets. This fact is evident from the table 4 where we see that only the *Tiling* and *Sequential* algorithms have converged on the **iris** datasets and table 5 which shows that only the *Perceptron Cascade*, *Tiling*, and *Sequential* algorithms have been successful on the **segmentation** dataset.

The modification of the pattern by appending an additional attribute also causes some

Algorithm	Network Size	Training Time	Test Accuracy
Cascade	20.96±2.72	490.67±106.98	74.43±2.15
Tiling	53.41±19.39	174.65±60.93	83.87±1.78
Sequential	29.48±2.18	1156.72±188.59	81.57±2.38

Table 5: Performance of the Constructive Algorithms on the **segmentation** dataset

practical difficulties. Certain real-world datasets have patterns with large magnitude attributes. Since the correctness proofs of the *Tower*, *Pyramid*, *Upstart*, and *Perceptron Cascade* algorithms require augmentation of the dataset with an additional attribute representing the sum of squares of the individual attributes, this additional attribute is often very large in magnitude. Such high magnitude attributes would cause an excruciating slow down in training the constructive algorithms. One solution to this problem is to normalize the patterns so that each pattern vector has a magnitude of 1. It must be noted that normalization transforms the pattern space. In the appendix (section A.1) we show how the convergence proofs of the constructive algorithms can be modified to deal with normalized pattern vectors.

In practice, the success of constructive learning algorithms is critically dependent on the performance of the TLU weight training method ($\mathcal{A}$). The close interaction between the network construction process and the training of individual TLUs is demonstrated by our experiments with the **wine** dataset. When the *Thermal perceptron algorithm* was used to play the role of $\mathcal{A}$ none of the constructive learning algorithms were able to converge. Replacing the *Thermal perceptron algorithm* by the *Barycentric correction procedure* produced entirely different results with all except the *Pyramid* algorithm converging to zero classification errors fairly quickly. These results are summarized in Table 6.

Algorithm	Network Size	Training Time	Test Accuracy
Tower	12.24±0.83	56.38±5.66	92.76±1.72
Upstart	14.76±10.17	491.18±505.99	90.48±3.76
Cascade	16.36±3.9	557.97±190.84	89.52±5.67
Tiling	7.56±0.51	24.54±1.43	93.04±3.67
Sequential	7.4±1.12	129.04±29.71	93.24±4.75

Table 6: Performance of the Constructive Algorithms on the **wine** dataset

Similarly, experiments with the **sonar** dataset revealed that a single TLU trained using the *Pocket algorithm* with *ratchet modification* could correctly classify the entire training set i.e., the dataset is linearly separable. Even after training a TLU for 1000 epochs

using the *Thermal perceptron algorithm* and the *Barycentric correction procedure* the separating weight vector was not found.

Another important factor which affects convergence of the constructive algorithms in the case of datasets with multiple output categories is the WTA training strategy. Tables 7 and 8 below summarize the performance of the constructive algorithms on the **iris** and the **segmentation** datasets using the WTA output strategy. We observe that for the **segmentation** dataset the *Upstart* algorithm converges using the WTA output strategy whereas its convergence using the independent output computation was not possible (see Table 5).

Algorithm	**Network Size**	**Training Time**	**Test Accuracy**
Tiling	8.0±0.0	29.17±0.99	95.92±0.7
Sequential	7.0±0.0	126.06±43.2	90.4±0.82

Table 7: WTA Output strategy on the **iris** dataset

Algorithm	**Network Size**	**Training Time**	**Test Accuracy**
Upstart	14.76±1.94	292.86±72.35	86.77±1.47
Cascade	13.88±1.13	269.12±44.26	86.79±1.52
Tiling	30.32±4.34	153.85±21.46	86.81±1.25
Sequential	30.16±3.2	1997.75±489.26	83.64±1.97

Table 8: WTA Output strategy on the **segmentation** dataset

8.2 Network Size

A major motivation for exploring constructive learning algorithms is their ability to generate parsimonious networks. The convergence proofs for constructive algorithms are existence proofs and are based on the ability of each added neuron to reduce the classification error by at least one. A trivial network construction process assigning one neuron per pattern would achieve zero classification errors. In this case, neither the network size nor the generalization performance of the resultant network would be satisfactory. We argue that in practice the algorithms we have presented perform much better. A comparison of the average network sizes (see Tables 1 — 8), in the cases where the networks generated actually converged to zero training errors, to the total size of the training set (see Table 10) demonstrates that the networks generated were compact in the sense that the constructive algorithms did not simply memorize the training patterns by assigning a single hidden node to classify each pattern.

A comparison of the average network sizes generated for the **segmentation** with and without the WTA output strategy (see Tables 5 and 8 respectively) shows one case wherein the WTA output strategy yields substantially smaller networks.

8.3 Generalization Performance

Although convergence and network size are important parameters of constructive algorithms, generalization is a more meaningful yardstick for measuring their performance. A single layer of TLUs when trained has a certain generalization ability. Of course, this single layer of TLUs cannot converge to zero classification errors in the case of non-linearly separable training sets. A constructive algorithm can generate a network with zero classification errors on non-linearly separable sets. However, in cases where the size of the training set is small or there is noise in the training data the use of the constructive algorithm might result in over-fitting. The added neurons might effectively memorize a few patterns misclassified by the first layer of TLUs. When this happens, the generalization performance of the resulting networks can be worse than that of the single layer network.

We present the results of training a single layer of TLUs (see table 9) using the *Thermal perceptron algorithm* to classify each dataset. A description of the training methodology is provided in the appendix (section A.3).

Dataset	**Training Accuracy**	**Training Time**	**Test Accuracy**
r5	56.37±2.54	3.26±0.08	—
r5 (WTA)	75.78±1.18	4.26±0.05	—
3 circles	23.11±9.76	102.40±2.00	22.26 ± 9.48
3 circles (WTA)	44.86±4.13	126.70±1.86	42.46 ± 4.20
Iris	78.36±0.95	8.17±0.1	72.64±1.7
Iris (WTA)	99.0±0.0	11.9±0.17	98.0±0.0
Segmentation	82.59±0.7	45.95±0.94	71.44±0.67
Segmentation (WTA)	94.31±0.57	50.46±0.84	87.39±0.42
Ionosphere	95.42±0.59	17.46±0.15	92.99±1.93

Table 9: Single Layer Training — *Thermal perceptron algorithm*

A significant increase in generalization performance is observed for the **3-circles** (see table 2), the **iris** (see table 4), and the **segmentation** (see table 5) datasets with independent training. The performance on the **ionosphere** dataset improved only marginally (see table 3). When the WTA training was employed the performance of the constructive algorithms on the **segmentation** and **iris** datasets actually deteriorated (see tables 7 and 8).

In summary, given adequate training data, constructive algorithms can yield relatively compact networks that significantly outperform the single layer networks for the same task in terms of generalization accuracy. However, in practice, it might be necessary to terminate the network construction algorithm before over-fitting sets in.

8.4 Training Speed

The issue of network training times becomes critical for very large training sets. We have measured the average training times for each dataset (see tables 1 — 9). Below we discuss some factors that affect the training times. We must point out that our simulation programs did not contain any special optimization beyond the facilities provided by the compiler and standard techniques for enhancing the run time performance of the programs.

A comparison of the average training times across different algorithms clearly shows that the *Tower*, *Pyramid*, and *Tiling* algorithms are able to learn relatively faster as compared to the *Upstart*, *Perceptron Cascade*, and *Sequential* algorithms. This can be explained in terms of the operational characteristics of the algorithms. The *Upstart* and *Perceptron Cascade* algorithms require re-training of the output weights after each daughter neuron is added and trained. This computation is fairly time consuming especially since the fan in of the output neurons increases with the addition of each new daughter. The *Sequential* learning algorithm is limited by the fact that the only suitable TLU weight training algorithm available to exclude patterns belonging to a single class is a variant of the *Barycentric correction procedure*. The multi-category extension of this procedure involves running the two-category version for each of the output classes which explains why the *Sequential* learning algorithm learns very slowly for pattern sets involving large number of output classes. Faster learning in the *Tower* and *Pyramid* is attributed to the fact that each layer of the network is trained just once and the weights are frozen. In the case of the *Tiling* network, in addition to the fact that the neurons are trained only once, the training set sizes for the ancillary neurons progressively decrease as additional ancillary neurons get added. Since each neuron or group of neurons are trained for 500 epochs irrespective of the training set size, smaller training sets obviously require less training time than larger ones. The same advantage holds for *Sequential* learning.

9 Summary and Discussion

Constructive algorithms offer an attractive approach to automated design of neural networks for pattern classification. In particular, they eliminate the need for ad hoc, and often inappropriate, a-priori choice of network architecture; potentially provide a means of constructing networks whose size (complexity) is commensurate with the complexity

of the pattern classification task at hand; and offer natural ways to incorporate prior knowledge (e.g., in the form of classification rules, decision trees, etc.) to guide learning. In this paper, we have focused on a family of such algorithms that incrementally construct networks of threshold neurons. Although a number of such algorithms have been proposed in the literature, most of them are limited to 2-category pattern classification tasks with binary/bipolar valued input attributes. This paper extends several existing constructive learning algorithms to handle multi-category classification for patterns having real-valued attributes. We have provided rigorous proofs of convergence to zero classification errors on finite, non-contradictory training sets for each of the multi-category algorithms proposed in this paper. Our proof technique provides a sufficiently general framework to prove the convergence of several different constructive algorithms. This strategy will be useful in proving the convergence properties of constructive algorithms designed in the future.

The convergence of the proposed algorithm to zero classification errors was established by showing that each modification of the network topology guarantees the existence of a weight setting that would yield a classification error that is less than that provided by the network before the modification and assuming a weight modification algorithm $\mathcal{A}$ that would find such a weight setting. We do not have a rigorous proof that any of the graceful variants of perceptron learning algorithms that can in practice, satisfy the requirements imposed on $\mathcal{A}$, let alone find an *optimal* (in some suitable well-defined sense of the term - e.g., so as to yield minimal networks) set of weights. The design of suitable TLU training algorithms that (with a high probability) satisfy the requirements imposed on $\mathcal{A}$ and are at least approximately optimal remains an open research problem. Against this background, the primary purpose of the experiments described in section 8 was to explore the actual performance of such multi-category constructive learning algorithms on some non-linearly separable classification tasks if we were to use a particular variant of perceptron learning for non-linearly separable datasets. Detailed theoretical and experimental analysis of the performance of single threshold neuron training algorithms is in progress [Yang *et al.*, 1997b]. We expect this analysis to lead to the design of improved and possibly hybrid weight modification schemes that can dynamically adapt to the situation faced by the particular constructive algorithm on a given dataset. For example, in certain pattern configurations it might be appropriate to exclude as many patterns of one class as possible whereas in other scenarios it might be better to correctly classify as large a subset of the training patterns as possible.

Simulation results have demonstrated the usefulness of the constructive algorithms in classification tasks. Some of the issues addressed in the preceding sections do set the stage for a detailed evaluation of the design choices that affect the performance of the constructive learning algorithms and identify several avenues for further research:

1. A cross-validation based criterion for training constructive networks must be employed wherein the training is stopped when the network's generalization begins

to deteriorate after the addition of a new neuron (or a group of neurons). It is likely to generate compact networks that exhibit good generalization properties with relatively little training as opposed to the current stopping-criterion of zero classification errors which might lead to over-fitting of the training set.

2. Hybrid network training schemes that dynamically select an appropriate network construction strategy, an appropriate TLU weight training algorithm, an appropriate output computation strategy and such to obtain locally optimal performance at each step of the classification task are likely to yield superior performance across a variety of datasets.

3. Various pre-processing techniques are responsible for transforming the training data in a manner that might simplify the learning task. Among these we have already seen the benefits of normalization. Another method of handling pattern sets with real-valued outputs is quantization of the training patterns. Preliminary results of applying quantization are presented in [Yang & Honavar, 1996].

4. Post-processing techniques such as *pruning* of networks eliminate nodes and connections that do not adversely affect the network's performance. Pruning can potentially overcome the over-fitting problem by yielding more compact networks with superior generalization. In recent work it was demonstrated that the application of simple pruning strategies on the *Tiling* networks leads to substantial reduction in the network sizes [Parekh *et al.*, 1997c].

5. The differences in the training times of the various constructive algorithms are striking. This may be due, among other things, to the differences in their inductive and representational biases. However, it might be possible in some cases to optimize each algorithm separately to reduce its training time.

6. Each constructive algorithm has its own set of inductive and representational biases implicit in the design choices that determine when and where a new neuron is added and how it is trained. A systematic characterization of this bias would be useful in guiding the design of constructive algorithms exhibiting improved performance.

7. It is often the case that the generalization performance of inductive learning algorithms can be substantially improved by augmenting them with suitable algorithms for selecting a relevant subset of a much larger set of input attributes many of which might be irrelevant or noisy. A variety of feature subset selection algorithms have been proposed in the literature on pattern recognition [Ripley, 1996]. The effectiveness of genetic algorithms for feature subset selection for pattern classification has been demonstrated by [Yang & Honavar, 1997]. Against this background, exploration of constructive learning algorithms augmented with suitable feature subset selection techniques might be of interest.

The impact of these factors on the training efficiency (network size and training time) and the generalization ability merit further investigation.

A systematic comparison of the performance of the constructive learning algorithms against the performance of simple classification schemes based on choice of appropriate distance metrics [Yang *et al.*, 1997a] and the popular backpropagation learning algorithm [Rumelhart *et al.*, 1986] would be useful in gaining a better understanding of their relative advantages and disadvantages. Other promising directions for further research include: incorporation of prior knowledge (e.g., in the form of rules) into the learning process, use of constructive algorithms in (sequential) cumulative multi-task learning wherein a single network is trained over a period of time to perform different tasks so that each task can exploit the useful regularities about the environment discovered by the network in the course of learning to perform the previous tasks.

A Appendix

In this section we discuss the convergence proof of the *Tower* algorithm given a normalized training set and describe the various datasets and training methodologies used in experiments with constructive algorithms.

A.1 Convergence Proof for the Tower Algorithm given Normalized Patterns

We show that if the dataset comprises of normalized patterns, convergence of the *Tower* algorithm is possible without making use of the extra attribute (that represents the sum of squares of the N pattern attributes). The original convergence proof for real-valued datasets is modified slightly in that the weight of the $N+1^{th}$ pattern input is added to the bias of the neuron and the additional input attribute (which is 1 for all the patterns in the normalized case) is dropped. It is easy to see that the proofs of the *Pyramid*, *Upstart*, and *Perceptron Cascade* algorithms can be obtained similarly from the corresponding proofs for the real-valued datasets seen earlier by making a similar modification as in the case of the *Tower* algorithm.

Theorem 7:
There exists a weight setting for neurons in the newly added layer L in the multi-category *Tower* network such that the number of patterns misclassified by the tower with L layers is less than the number of patterns misclassified prior to the addition of the L^{th} layer (i.e., $\forall L > 1,\ e_L < e_{L-1}$).

Proof:
Assume that all the patterns in the dataset are normalized. i.e., $\forall p\ \mathbf{X}^p$ is such that $\sum_{i=1}^{N}(X_i^p)^2 = 1$. Define $\kappa = \max_{p,q}\sum_{i=1}^{N}(X_i^p - X_i^q)^2$. For each pattern $\mathbf{X}^p$, define ϵ_p as $0 < \epsilon_p < \min_{p,q\neq p}\sum_{i=1}^{N}(X_i^p - X_i^q)^2$. It is clear that $0 < \epsilon_p < \kappa$ for all patterns $\mathbf{X}^p$. Assume that a pattern $\mathbf{X}^p$ was not correctly classified at layer $L-1$ (i.e., $\mathbf{C}^p \neq \mathbf{O}_{L-1}^p$). Consider the following weight setting for the neuron L_j.

$$\begin{aligned} W_{L_j,0} &= C_j^p(\kappa + \epsilon_p - \sum_{i=1}^{N}(X_i^p)^2) - C_j^p \\ W_{L_j,I_i} &= 2C_j^p X_i^p \text{ for } i = 1 \ldots N \\ W_{L_j,L-1_j} &= \kappa \\ W_{L_j,L-1_k} &= 0 \text{ for } k = 1 \ldots M, k \neq j \end{aligned} \tag{15}$$

In a manner analogous to equation (2) we see that the for the pattern $\mathbf{X}^p$ the net input

of neuron L_j is

$$\begin{aligned} n^p_{L_j} &= W_{L_j,0} + \sum_{i=1}^{N} W_{L_j,I_i} X^p_i + \sum_{i=1}^{M} W_{L_j,L-1_i} O^p_{L-1_i} \\ &= C^p_j(\kappa + \epsilon_p - \sum_{i=1}^{N}(X^p_i)^2) - C^p_j + 2C^p_j \sum_{i=1}^{N}(X^p_i)^2 + \kappa O^p_{L-1_j} \\ &= C^p_j(\kappa + \epsilon_p) + \kappa O^p_{L-1_j} \text{ since } \sum_{i=1}^{N}(X^p_i)^2 = 1 \end{aligned} \tag{16}$$

If $C^p_j = -O^p_{L-1_j}$:

$$\begin{aligned} n^p_{L_j} &= C^p_j \epsilon_p \\ O^p_{L_j} &= sgn(n^p_{L_j}) \\ &= C^p_j \text{ since } \epsilon_p > 0 \end{aligned}$$

If $C^p_j = O^p_{L-1_j}$:

$$\begin{aligned} n^p_{L_j} &= (2\kappa + \epsilon_p) C^p_j \\ O^p_{L_j} &= sgn(n^p_{L_j}) \\ &= C^p_j \text{ since } \kappa, \epsilon_p > 0 \end{aligned}$$

Thus we have shown that the pattern $\mathbf{X}^p$ is corrected at layer L. Now consider a pattern $\mathbf{X}^q \neq \mathbf{X}^p$. Note that for normalized patterns $-C^p_j$ can be re-written as $-C^p_j(\sum_{i=1}^{N}(X^q_i)^2)$.

$$\begin{aligned} n^q_{L_j} &= W_{L_j,0} + \sum_{i=1}^{N} W_{L_j,I_i} X^q_i + \sum_{i=1}^{M} W_{L_j,L-1_i} O^q_{L-1_i} \\ &= C^p_j(\kappa + \epsilon_p - \sum_{i=1}^{N}(X^p_i)^2) - C^p_j + 2C^p_j \sum_{i=1}^{N}(X^p_i)(X^q_i) + \kappa O^q_{L-1_j} \\ &= C^p_j(\kappa + \epsilon_p) + \kappa O^q_{L-1_j} - C^p_j \sum_{i=1}^{N}[(X^p_i)^2 - 2(X^p_i)(X^q_i) + (X^q_i)^2] \\ &= C^p_j(\kappa + \epsilon_p) + \kappa O^q_{L-1_j} - C^p_j [\sum_{i=1}^{N}(X^p_i - X^q_i)^2] \\ &= C^p_j(\kappa + \epsilon_p - \epsilon') + \kappa O^q_{L-1_j} \text{ where } \epsilon' = \sum_{i=1}^{N}(X^p_i - X^q_i)^2; \text{ note } \epsilon' > \epsilon_p \\ &= \kappa' C^p_j + \kappa O^q_{L-1_j} \text{ where } \kappa + \epsilon_p - \epsilon' = \kappa' \\ O^q_{L_j} &= sgn(n^q_{L_j}) \\ &= O^q_{L-1_j} \text{ since } \kappa' < \kappa \end{aligned} \tag{17}$$

Thus, for all patterns $\mathbf{X}^q \neq \mathbf{X}^p$, the outputs produced at layers L and $L-1$ are identical. We have shown the existence of a weight setting that is guaranteed to yield a reduction

in the number of misclassified patterns whenever a new layer is added to the *Tower* network. We rely on the TLU weight training algorithm $\mathcal{A}$ to find such a weight setting. Since the training set is finite in size, eventual convergence to zero errors is guaranteed. □

The proof of convergence for the *Tower* algorithm with the WTA output can be reworked similarly for normalized patterns.

A.2 Datasets

A cross-section of datasets having real-valued pattern attributes and patterns belonging to multiple output classes was selected for the simulation experiments. Table 10 summarizes the characteristics of the datasets. **Train** and **Test** denote the size of the training and test sets respectively. **Inputs** indicates the total number of input attributes (of the unmodified dataset). **Outputs** represents the number of output classes while **Attributes** describes the type of input attributes of the patterns. The real-world datasets **ionosphere**, **iris**, **segmentation**, and **wine** are available at the UCI Machine Learning Repository [Murphy & Aha, 1994] while the **sonar** dataset is taken from the CMU Artificial Intelligence Repository[7]. Given that the **segmentation** and **wine** datasets involve attributes with high magnitudes, we used normalized versions of these datasets.

Dataset	**Train**	**Test**	**Inputs**	**Outputs**	**Attributes**
5 bit random (**r5**)	32	0	5	3	bipolar
3 concentric circles (**3 circles**)	900	900	2	3	real
ionosphere structure (**ionosphere**)	234	117	34	2	real, int
image segmentation (**segmentation**)	210	2100	19	7	real, int
iris plant (**iris**)	100	50	4	3	real
wine recognition (**wine**)	120	58	13	3	real, int
sonar (**sonar**)	104	104	60	1	real

Table 10: Datasets

A.3 Training Methodology

Any of the three TLU weight training schemes can fit the role of $\mathcal{A}$ for the *Tower Pyramid*, *Tiling*, *Upstart*, and *Perceptron Cascade* algorithms. Initially, the *Thermal perceptron algorithm* was used for training weights of the individual TLUs. The weights of each neuron were randomly initialized to values between -1 and $+1$. The number of training epochs was set to 500. Each epoch involves presenting a set of l randomly drawn patterns from the training set where l is the size of the training set. The initial temperature T_0 was set to 1.0 and was dynamically updated at the end of each epoch to match the average net input of the neuron(s) during the entire epoch [Burgess, 1994]. 25

[7]URL — http://www.cs.cmu.edu/Groups/AI/html/repository.html

runs were conducted for each experimental set up. Training was stopped if the network failed to converge to zero classification errors after adding either 100 hidden neurons in a given layer or after training a total of 25 hidden layers and that particular run was designated as a failure. The single layer networks of TLUs (see table 9) were trained for 25 runs using 500 epochs of the *Thermal perceptron algorithm*.

In *Sequential* learning only, the variation of the *Barycentric correction procedure* which is specifically designed for exclusion of patterns can be used for training. Each hidden neuron was trained for 500 epochs of the *Barycentric correction procedure* with the initial weighting coefficients set to random values between 1 and 3.

In the case of the *Upstart* and *Perceptron Cascade* algorithms, some runs failed to converge to zero classification errors. Upon closer scrutiny it was found that the training sets of the daughter neurons had very few patterns with a target output of 1. The *Thermal perceptron algorithm* while trying to correctly classify the largest subset of training patterns ended up assigning an output of 0 to all patterns. Thus it failed to meet the requirements imposed on $\mathcal{A}$ in this case. This resulted in the added daughter neuron's failure to reduce the number of misclassified patterns by at least one and in turn caused the *Upstart* and the *Perceptron Cascade* algorithms to keep adding daughter neurons without converging. To overcome this problem, a *balancing* of the training set for the daughter neuron was performed as follows: The daughter neuron's training set was balanced by replicating the patterns having target output 1 sufficient number of times so that the dataset has the same number of patterns with target 1 as with target 0. Given the tendency of the *Thermal perceptron algorithm* to find a set of weights that correctly classify a near-maximal subset of its training set, it was now able to (with the modified training set) at least approximately satisfy the requirements imposed on $\mathcal{A}$.

References

Burgess, N. (1994). A Constructive Algorithm That Converges for Real-Valued Input Patterns. *International Journal of Neural Systems*, **5**(1), 59–66.

Chen, C-H., Parekh, R., Yang, J., Balakrishnan, K., & Honavar, V. (1995). Analysis of Decision Boundaries generated by Constructive Neural Network Learning Algorithms. *Pages 628–635 of: Proceedings of WCNN'95, July 17-21, Washington D.C.*, vol. 1.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Fahlman, S. E., & Lebiere, C. (1990). The Cascade Correlaion Learning Architecture. *Pages 524–532 of:* Touretzky, D. S. (ed), *NIPS*.

Frean, M. (1990a). *Small Nets and Short Paths: Optimizing Neural Computation*. Ph.D. thesis, Center for Cognitive Science, Edinburgh University, UK.

Frean, M. (1990b). The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, **2**, 198–209.

Frean, M. (1992). A Thermal Perceptron Learning Rule. *Neural Computation*, **4**, 946–957.

Gallant, S. (1990). Perceptron Based Learning Algorithms. *IEEE Transactions on Neural Networks*, **1**(2), 179–191.

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

Garey, M., & Johnson, D. (1979). *Computers and Intractability*. New York: W. H. Freeman.

Honavar, V. (1990). *Generative Learning Structures and Processes for Generalized Connectionist Networks*. Ph.D. thesis, University of Wisconsin, Madison.

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

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

Marchand, M., Golea, M., & Rujan, P. (1990). A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons. *Europhysics Letters*, **11**(6), 487–492.

Mézard, M., & Nadal, J. (1989). Learning Feed-forward Networks: The Tiling Algorithm. *J. Phys. A: Math. Gen.*, **22**, 2191–2203.

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press.

Murphy, P., & Aha, D. (1994). *UCI Repository of Machine Learning Databases.* Department of Information and Computer Science, University of California, Irvine, CA. [http://www.ics.uci.edu/AI/ML/MLDBRepository.html].

Nilsson, N.J. (1965). *The Mathematical Foundations of Learning Machines.* New York: McGraw-Hill.

Parekh, R., Yang, J., & Honavar, V. (1995). *Constructive Neural Network Learning Algorithms for Multi-Category Classification.* Tech. rept. ISU-CS-TR95-15a. Department of Computer Science, Iowa State University.

Parekh, R. G., Yang, J., & Honavar, V. G. (1997a). *An Experimental Comparison of Constructive Neural Network Learning Algorithms.* In preparation.

Parekh, R. G., Yang, J., & Honavar, V. G. (1997b). MUpstart - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. *In: Proceedings of the IEEE/INNS International Conference on Neural Networks, ICNN'97.* To appear.

Parekh, R. G., Yang, J., & Honavar, V. G. (1997c). Pruning Strategies for Constructive Neural Network Learning Algorithms. *In: Proceedings of the IEEE/INNS International Conference on Neural Networks, ICNN'97.* To appear.

Poulard, H. (1995). Barycentric Correction Procedure: A Fast Method of Learning Threshold Units. *Pages 710–713 of: Proceedings of WCNN'95, July 17-21, Washington D.C.*, vol. 1.

Ripley, B. (1996). *Pattern Recognition and Neural Networks.* New York: Cambridge University Press.

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., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. *In: Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, vol. 1 (Foundations). Cambridge, Massachusetts: MIT Press.

Saffery, J., & Thornton, C. (1991). Using Stereographic Projection as a Preprocessing Technique for Upstart. *Pages II 441–446 of: Proceedings of the International Joint Conference on Neural Networks.* IEEE Press.

Siu, K-Y., Roychowdhury, V., & Kailath, T. (1995). *Discrete Neural Computation - A Theoretical Foundation.* Englewood Cliffs, NJ: Prentice-Hall.

Yang, J., & Honavar, V. (1996). A Simple Randomized Quantization Algorithm for Neural Network Pattern Classifiers. *Pages 223–228 of: Proceedings of the World Congress on Neural Networks'96.*

Yang, J., & Honavar, V. (1997). *Feature Subset Selection Using a Genetic Algorithm.* Tech. rept. ISU-CS-TR 97-02. Iowa State University.

Yang, J., Parekh, R., & Honavar, V. (1996). MTiling - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. *Pages 182–187 of: Proceedings of the World Congress on Neural Networks'96.*

Yang, J., Parekh, R., & Honavar, V. (1997a). DistAI*: An Inter-pattern Distance-based Constructive Learning Algorithm.* Tech. rept. ISU-CS-TR 97-05. Iowa State University.

Yang, J., Parekh, R., & Honavar, V. (1997b). *Empirical Comparison of Graceful Variants of the Perceptron Learning Algorithm on Non-Separable Data Sets.* In preparation.