Constructive Neural Network
Learning Algorithms for
Multi-Category Pattern Classification

TR #95-15a

Rajesh Parekh, Jihoon Yang, and Vasant Honavar

December 5, 1995

This revised version supersedes ISU CS Technical Report TR 95-15.


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

Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040, USA
Constructive Neural Network Learning Algorithms for Multi-Category 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

December 5, 1995

Abstract

Constructive learning algorithms offer an approach for incremental construction of potentially near-minimal neural network architectures for pattern classification tasks. Such 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 otherwise 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 a finite, non-contradictory training set in a 2-category classification problem. This paper explores multi-category extensions of several constructive neural network learning algorithms for pattern classification. In each case, we establish the convergence to zero classification errors on a multi-category classification task (under certain assumptions). Results of experiments with non linearly separable multi-category data sets demonstrate the feasibility of this approach to multi-category pattern classification and also suggest several interesting directions for future research.

*This research was partially supported by the National Science Foundation grant IRI-9409580 to Vasant Honavar.
1 Introduction

Multi-layer networks of threshold logic units (TLU) or multi-layer perceptrons (MLP) offer a particularly attractive framework for the design of pattern classification 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, 93]; and simpler digital hardware realizations than their continuous counterparts.

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 unit that computes a 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 \( W \), in response to a pattern \( X^p \), is a bipolar hardlimiting function of \( W \cdot X^p \), i.e.

\[
O^p = \begin{cases} 
1 & \text{if } W \cdot X^p > 0 \\
-1 & \text{otherwise}
\end{cases}
\]

Such a TLU or threshold neuron implements a \( (N - 1) \)-dimensional hyperplane given by \( W \cdot X = 0 \) 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 \( S_+ = \{(X^p, C^p) \mid C^p = 1\} \) and \( S_- = \{(X^p, C^p) \mid C^p = -1\} \) \( (C^p \) is the desired output of the pattern classifier for the input pattern \( X^p \)), it is the goal of a perceptron training algorithm to attempt find a weight vector \( \hat{W} \) such that \( \forall X^p \in S_+, \hat{W} \cdot X^p > 0 \) and \( \forall X^p \in S_-, \hat{W} \cdot X^p \leq 0 \). If such a weight vector \( \hat{W} \) exists for the pattern set \( S \) then \( S \) is said to be linearly separable. Several iterative algorithms are available for finding such a \( \hat{W} \) if one exists [Nilsson, 65; Duda & Hart, 73]. Most of these are variants of the perceptron weight update rule: \( W \leftarrow W + \eta (C^p - O^p)X^p \) (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 [Gallant, 93], thermal perceptron [Frean, 90], loss minimization algorithm [Hrycej, 92], and the barycentric correction procedure [Poulard, 95] 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 converge to zero classification errors when \( S \) is linearly separable. For a detailed comparison of the single TLU training algorithms see [Yang et al., 95]. Recently [Siu et al., 95] have established the necessary and sufficient conditions for a training set \( S \) to be non-linearly separable. They have also uncovered structures within a non-linearly separable set \( S \) and have shown that the problem of identifying a largest linearly separable subset \( S_{\text{sep}} \) 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 or the thermal perceptron to correctly classify as large a subset of training
patterns as possible within the given constraints (such as limited training time).

When $S$ is not linearly separable, however, a multi-layer network of TLUs is needed to learn a complex 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) training set. Some of the motivations for studying such algorithms [Honavar, 90; Honavar & Uhr, 93] include:

- **Limitations of learning by weight modification alone within an otherwise 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 (as measured 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. And 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 data set.

- **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 statistical distribution 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 subsets of performance measures (e.g., learning time) against others (network size, generalization accuracy).

- **Incorporation of prior knowledge:** Constructive algorithms provide a natural framework for exploiting problem-specific knowledge (e.g., in the form of production
rules) into the initial network configuration or heuristic knowledge (e.g., about the general topological constraints on the network) into the network construction algorithm.

A number of constructive 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, 90], tiling [Mézard & Nadal, 89], upstart [Frean, 90], and perceptron cascade [Burgess, 91]. They are all based on the idea of transforming the hard 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 (e.g., the pocket algorithm)

Different 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, 95] 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 in the 2-category case. The convergence proof in each case is based on the ability of the variant of the perceptron 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 unit (or a set of units) is added and trained. We will refer to such a variant of the perceptron algorithm as $L_W$. In practice, the performance of the constructive algorithm depends partly on the choice of $L_W$ and its ability to find weight settings that reduce the total number of misclassifications each time a new unit is added to the network and trained. Some possible choices for $L_W$ are the pocket algorithm, the thermal perceptron, and other variants of the perceptron algorithm for non linearly separable data sets.

Pattern classification tasks that arise in practice 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 $X^p$ in category $\Psi_i$ can be used by the learning algorithm to effectively rule out its membership in a different category $\Psi_j$ ($j \neq i$) and any internal representations learned in inducing the structure of $\Psi_i$ can therefore be
exploited in inducing the structure of a category $\Psi_j$ ($j \neq i$). Thus, extensions of 2-category constructive learning algorithms to deal with multi-category classification tasks are clearly of interest. However, in most cases, such extensions have not been explored while 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. 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. The rest of the paper is organized as follows: Section 2 explores multi-category extensions of tower, pyramid, upstart, tiling, and perceptron cascade algorithms. In each case, convergence to zero classification errors is established. Section 3 presents some preliminary results on two classification tasks (an artificial task involving random boolean mappings, and a real-world task of classifying the iris data set). Section 4 concludes with a summary and discussion of some directions for future research.

2 Multi-Category Constructive Learning Algorithms

This section outlines several multi-category constructive learning algorithms. Some of these are relatively straightforward extensions of the corresponding 2-category algorithms whereas others entail non-trivial modifications and present several interesting design choices. Proof of convergence to zero classification errors on any given finite, non-contradictory training set is provided in each case.

2.1 Notation

The following notation is used in the convergence proofs of the constructive learning algorithms.

Number of input neurons: $N$
Number of output neurons (equal to the number of categories): $M$
Categories: $\Psi_1, \Psi_2, \ldots, \Psi_M$
Number of units in layer $A$: $U_A$
Weight vector for neuron $j$: $W_j$
Net Input for neuron $j$ of layer $A$ in response to pattern $X^p$: $n^p_{A,j}$
Threshold (or bias) for unit $i$ of layer $A$: $W_{A,i,0}$
Connection weight between unit $i$ of layer $A$ and unit $j$ of layer $B$: $W_{A,i,B,j}$
Indexing for neurons of layer $A$: $A_1, A_2, \ldots, A_{U_A}$
For the input layer $A = I$
Augmented Pattern vector $p$: $X^p = < X_0^p, X_1^p, \ldots, X_N^p >$, $X_0^p = 1$ for all $p$
Target output for pattern $X^p$: $C^p = < C_1^p, C_2^p, \ldots, C_M^p >$, $C_i^p = 1$ if $X^p \in \Psi_i$ and $C_i^p = -1$ otherwise
Observed output for pattern $X^p$ at layer $A$: $O_A^p = < O_{A_1}^p, O_{A_2}^p, \ldots, O_{A_k}^p >$ where $U_A = k$
A pattern is said to be correctly classified at layer $A$ when $C^p = O^p_A$.

Number of patterns wrongly classified at layer $A$: $e_A$

A function $sgn$ is defined as $sgn(x) = -1$ if $x < 0$ and $sgn(x) = 1$ if $x \geq 0$ where $x$ is a real number.

2.2 Tower Algorithm

The 2-category Tower algorithm [Gallant, 90] constructs a tower of TLUs. The bottom-most neuron in the tower receives $N$ inputs, one for each component of the pattern vector. The tower is built by successively adding neurons to the network and training them using $L_W$ until the desired classification accuracy is achieved. Each newly added neuron becomes the new output neuron and receives as input each of the $N$ components of the input pattern as well as the output of the neuron immediately below itself.

The extension of the 2-category tower algorithm to deal with multiple ($M$) output categories is rather straightforward. It can be accomplished by simply adding $M$ neurons each time a new layer is added to the tower. Each neuron in the newly added layer (which becomes the new output layer) receives inputs from the $N$ input neurons as well as the $M$ neurons in the preceding layer. The topology of the resulting multi-category tower network is shown in Fig. 1.

2.2.1 Multi-Category Tower Algorithm

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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{tower_network.png}
\caption{Tower Network}
\end{figure}
2. Repeat the following steps until the desired training accuracy is achieved or the maximum number of hidden layers allowed is exceeded.

3. $L = L + 1$. 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 each of the input neurons and to each neuron in the preceding layer $L-1$ (if one exists).

4. Train the weights associated with each of the newly added neurons in layer $L$ (the rest of the weights in the network are left unchanged).

2.2.2 Convergence Proof

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

Proof:
Assume that a pattern $X^p$ was not correctly classified at layer $L - 1$ (i.e. $C^p \neq O^p_{L-1}$). Consider the following weight setting for the neuron $j$ ($j = 1 \ldots M$) in layer $L$ (the newly added output layer).

$$
W_{Lj,0} = C^p_j
$$

$$
W_{Lj,i} = C^p_j \cdot X^p_i \text{ for } i = 1 \ldots N
$$

$$
W_{Lj,L-1} = N
$$

$$
W_{Lj,L-1,k} = 0 \text{ for } k = 1 \ldots M, k \neq j
$$

Input Layer Connections

Connections to Layer L-1

![Figure 2: Weight Setting for the jth output neuron in the Tower Network](image)

For the pattern $X^p$ the net input $n^p_{Lj}$ for the $j$th unit in layer $L$ is:

$$
n^p_{Lj} = W_{Lj,0} + \sum_{i=1}^{i=N} W_{Lj,i}X^p_i + \sum_{i=1}^{i=M} W_{Lj,L-1}O^p_{L-1,i}
$$
\[ = C_p^j + C_p^j N + NO^p_{L-1} \]

If \( C_p^j = -O^p_{L-1} \):

\[
\begin{align*}
n^p_{L,j} &= C_p^j \\
o^p_{L,j} &= \text{sgn}(n^p_{L,j}) \\
&= C_p^j
\end{align*}
\]

If \( C_p^j = O^p_{L-1} \):

\[
\begin{align*}
n^p_{L,j} &= (2N + 1)C_p^j \\
o^p_{L,j} &= \text{sgn}(n^p_{L,j}) \\
&= C_p^j
\end{align*}
\]

Thus we have shown that the pattern \( X^p \) is corrected at layer \( L \). Now consider a pattern \( X^q \neq X^p \). Clearly, \( X^p \cdot X^q \leq N - 2 \) for bipolar patterns.

\[
\begin{align*}
n^q_{L,j} &= W_{L,j,0} + \sum_{i=1}^{N} W_{L,j,i} \cdot X^q_i + \sum_{i=1}^{M} W_{L,j,L-1,i} \cdot O^q_{L-1,i} \\
&= C_p^j + C_p^j X^p \cdot X^q + NO^q_{L-1} \\
&\leq (N - 1)C_p^j + NO^q_{L-1} \\
o^q_{L,j} &= \text{sgn}(n^q_{L,j}) \\
&= O^q_{L-1}
\end{align*}
\]

Thus, for all patterns \( X^q \neq X^p \), 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. We rely on the algorithm \( L_W \) to find such a weight setting. Since the training set is finite in size, eventual convergence to zero errors is guaranteed.

\[ \square \]

### 2.3 Pyramid Algorithm

The 2-category pyramid algorithm [Gallant, 90] 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 at each of the preceding layers. The newly added neuron constitutes the new output of the network. As in the case of the tower algorithm, the extension of the 2-category tower algorithm to handle \( M \) output categories is quite straightforward with each newly added layer of \( M \) neurons receiving inputs from the \( N \) input neurons and the outputs of each neuron in each of the previously added layers. The resulting \( M \)-category tower network is shown in Fig. 3.
2.3.1 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.

3. $L = L + 1$. 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 $N$ inputs and each neuron in each of the previous layers.

4. Train the weights associated with each of the newly added neurons in layer $L$ (the rest of the weights in the network are left unchanged).

2.3.2 Convergence Proof

Theorem 2:
There exists a weight setting for neurons in the newly added layer $L$ in the $M$-category
pyramid network such that the number of patterns misclassified by the pyramid with \( L \) layers is less than the number of patterns misclassified by the same pyramid prior to the addition of the \( L \)th layer (i.e., \( \forall L > 1, \quad \epsilon_L < \epsilon_{L-1} \)).

**Proof:**

Assume that a pattern \( X^p \) was not correctly classified in layer \( L - 1 \) (i.e. \( C^p \neq O^p_{L-1} \)). Consider the following weight setting for neuron \( j (j = 1 \ldots M) \) in layer \( L \) (the newly added output layer).

\[
\begin{align*}
W_{Lj,0} &= C^p_j \\
W_{Lj,L_i} &= C^p_j X^p_i \quad \text{for} \quad i = 1 \ldots N \\
W_{Lj,L-i_k} &= 0 \quad \text{for} \quad i = 2 \ldots L - 1, \quad \text{and} \quad k = 1 \ldots M \\
W_{Lj,L-1} &= N \\
W_{Lj,L-1_k} &= 0 \quad \text{for} \quad k = 1 \ldots M, k \neq j
\end{align*}
\]

![Figure 4: Weight Setting for the jth output neuron in the Pyramid Network](image)

This choice of weights reduces an \( M \)-category pyramid network to an \( M \)-category tower network. The convergence proof follows directly from the convergence proof of the tower algorithm.

\[ \square \]

### 2.4 Upstart Algorithm

The 2-category upstart algorithm [Frean, 90] 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 may exist between the different \( M \)-outputs. We therefore follow an alternative approach (suggested by Frean) 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 TLU's implementing the binary hardlimiter function, we will present our extension of this algorithm to \( M \) classes under the same binary valued
framework.\footnote{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.}

First, an output layer of $M$ neurons is trained using the chosen $L_W$ algorithm. 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 ($k$) that makes the most number of errors (in the sense $C^p_k \neq O^p_k$) is identified. Depending upon whether the neuron $k$ is wrongly-on (i.e. $C^p_k = 0, O^p_k = 1$) or wrongly-off (i.e. $C^p_k = 1, O^p_k = 0$) 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 the output neuron $k$. For each pattern $X^p$ in the training set, the target outputs ($C^p_X$ and $C^p_Y$) for the $X$ and $Y$ daughters are determined as follows:

- If $C^p_k = 0$ and $O^p_k = 0$ then $C^p_X = 0, C^p_Y = 0$.
- If $C^p_k = 0$ and $O^p_k = 1$ then $C^p_X = 1, C^p_Y = 0$.
- If $C^p_k = 1$ and $O^p_k = 0$ then $C^p_X = 0, C^p_Y = 1$.
- If $C^p_k = 1$ and $O^p_k = 1$ then $C^p_X = 0, C^p_Y = 0$.

The daughter is trained using the $L_W$ algorithm, and after connecting it to each of the $M$ output units the output weights are retrained. The resulting network is shown in Fig. 5.

2.4.1 Multi-Category Upstart Algorithm

1. Train a single layer network with $M$ output units and $N$ input units using the algorithm $L_W$.

2. If the desired training accuracy is not achieved so 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 unit $k$ in the output layer that makes the most errors.
(b) Add a $X$ or a $Y$ daughter depending on whether the unit $k$ is wrongly-on or wrongly-off more often. The daughter unit is connected to all the $N$ inputs.
(c) Construct the training set for the daughter unit as described above and train it. Freeze the weights of this newly added daughter.
(d) Connect the daughter unit to each of the output neurons and retrain the output weights.

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.
2.4.2 Convergence 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 $M$ units$^2$. Thus far, the hidden layer comprises of $U_{L-1}$ daughter units. Assume also that the output neuron $z$ ($1 \leq z \leq M$) is wrongly on (i.e., it produces an output of 1 when the desired output is in fact 0) for a training pattern $X^p$. A $X$ daughter unit is added to the hidden layer and trained so as to correct the classification of $X^p$ at the output layer. The daughter unit is trained to output 1 for pattern $X^p$, and to output 0 for all other patterns. Next the newly added daughter unit is connected to all output units and the output weights are retrained.

**Theorem 3:**

There exists a weight setting for the $X$ daughter unit and the output units that ensures that the number of misclassified patterns is reduced by at least one for the multi-category upstart network.

**Proof:**

$^2$In the case of the multi-category 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 $L - 1 = 1$
Consider the following weight setting for the daughter unit:

\[
W_{X,0} = -\sum_{i=1}^{N} X_i^p \\
W_{X,i} = (2X_i^p - 1) \text{ for } i = 1 \ldots N
\]

For pattern \(X^p\):

\[
n_{X}^p = -\sum_{i=1}^{N} X_i^p + \sum_{k=1}^{N} W_{X,i_k} X_k^p \\
= -\sum_{i=1}^{N} X_i^p + \sum_{i=1}^{N} (2X_i^p - 1)X_i^p \\
= -\sum_{i=1}^{N} X_i^p + \sum_{i=1}^{N} X_i^p \text{ since } X_i^p \text{ can be either 0 or 1} \\
= 0 \\
O_{X}^p = 1 \text{ by definition of the threshold function}
\]

For any other pattern \(X^q \neq X^p\)

\[
n_{X}^q = -\sum_{i=1}^{N} X_i^q + \sum_{k=1}^{N} W_{X,i_k} X_k^q \\
= -\sum_{i=1}^{N} X_i^q + \sum_{i=1}^{N} (2X_k^q - 1)X_k^q \\
= -\sum_{i=1}^{N} X_i^q + 2\sum_{k=1}^{N} X_k^q X_k^q - \sum_{k=1}^{N} X_k^q
\]

Since \(X^q \neq X^p\), the number of attributes in which both \(X^q\) and \(X^p\) have a 1 is clearly less than the sum total number of 1’s in \(X^q\) and \(X^p\).

\[
n_{X}^q < 0 \\
O_{X}^q = sgn(n_{X}^q) \\
= 0
\]

Let \(\lambda_j = abs(W_{L_j,0}) + \sum_{k=1}^{N} abs(W_{L_j,i_k}) + \sum_{k=1}^{j-1} abs(W_{L_j,L-1_k}) \) (i.e. for each output unit \(j\), \(\lambda_j\) is the sum of absolute values of all its existing weights). Consider the following weight setting for connections between each output layer neuron and the newly trained \(X\) daughter:

\[
W_{L_j,X} = 2(C_j^p - O_j^p)\lambda_j
\]
$O^p_j$ is the original output of neuron $j$ in the output layer. Let us consider the new output of each neuron $j$ in the output layer in response to pattern $X^p$

$$n^p_{L_j} = W_{L_j,0} + \frac{1}{\min_{i=1}^{N}} W_{L_j, i} X^p_i + \sum_{k=U_{L-1}}^{k=U_{L-1}} W_{L_j, l} O^p_k + 2(C^p_j - O^p_j) \lambda_j O^p_X$$

$$= W_{L_j,0} + \sum_{i=1}^{N} W_{L_j, l} X^p_i + \sum_{k=U_{L-1}}^{k=U_{L-1}} W_{L_j, l} O^p_k + 2(C^p_j - O^p_j) \lambda_j \lambda_j (1)$$

We know that $\lambda_j \leq L_{L_j,0} + \sum_{i=1}^{N} W_{L_j, l} X^p_i + \sum_{k=U_{L-1}}^{k=U_{L-1}} W_{L_j, l} O^p_k \leq \lambda_j$.

- If $C^p_j = O^p_j$ we see that the net input for unit $L_j$ remains the same as that before adding the daughter unit and hence the output remains the same i.e., $C^p_j$.

- If $C^p_j = 0$ and $O^p_j = 1$, the net input for unit $L_j$ is $n^p_{L_j} \leq \lambda_j - 2 \lambda_j$. Since $\lambda_j \geq 0$, the new output of $L_j$ is 0 which is $C^p_j$.

- If $C^p_j = 1$ and $O^p_j = 0$, the net input for unit $j$ is $n^p_{L_j} \geq -\lambda_j + 2 \lambda_j$. Since $\lambda_j \geq 0$, the new output of $L_j$ is 1 which is $C^p_j$.

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

$$n^q_{L_j} = W_{L_j,0} + \sum_{i=1}^{N} W_{L_j, l} X^q_i + \sum_{k=U_{L-1}}^{k=U_{L-1}} W_{L_j, l} O^p_k + 2(C^p_j - O^p_j) \lambda_j O^p_X$$


\[ W_{L_j,0} + \sum_{i=1}^{N} W_{L_j,1_i} X_i^q + \sum_{k=1}^{k=U_{L-1}} W_{L_j,1_k} O_k^q \]

We see that the daughter’s contribution to the output neurons in the case of any patterns other than \( X^p \) 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 unit and hence the outputs for patterns other than \( X^p \) 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.

\[ \text{\Box} \]

### 2.5 Perceptron Cascade Algorithm

The perceptron cascade algorithm\(^3\) [Burgess, 94] 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, 90]. However, unlike the cascade correlation algorithm, the perceptron cascade algorithm uses TLUs. Initially an output neuron is trained using the \( L_W \) algorithm. If the output unit does not correctly classify the desired fraction of the training set, a daughter neuron is added and trained to correct some of the errors made by the output neuron. The daughter neuron receives inputs from each of the input units and from each of the previously added daughters. The targets for the daughter are determined exactly as in the case of the upstart network.

The extension to \( M \) output classes is relatively straight forward. First, the output layer of \( M \) neurons is trained. If the desired training accuracy is not achieved, the output neuron, \( k \), that makes the largest number of errors (in the sense that \( C_k^p \neq O_k^p \)) is identified and a daughter unit (an \( X \) daughter if the unit is wrongly-on more often or a \( Y \) daughter if the unit is wrongly-off more often) is added to the hidden layer and trained to correct some errors at the output layer. For each pattern \( X^p \) in the training set, the target outputs for the daughter unit 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 units and the output weights are retrained. Fig. 7 shows the construction of a perceptron cascade network.

\(^3\)Although the original two category version of this algorithm is guaranteed to converge for real valued patterns we have restricted our extension to multiple output classes to binary valued patterns only. The extensions to bipolar valued patterns and real valued patterns are straightforward.
2.5.1 Multi-Category Perceptron Cascade Algorithm

1. Train a single layer network with $M$ output units and $N$ input units using the algorithm $L_W$.

2. If the desired training accuracy is not achieved so 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 unit) is exceeded.

   (a) Determine the unit $k$ in the output layer 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 the unit $k$ is wrongly-on or wrongly-off more often. The daughter unit is connected to all the $N$ inputs and to all previously added
daughter units.

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

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

2.5.2 Convergence Proof

**Theorem 4:**
There exists a weight setting for each daughter unit added and the output units that ensures that the number of misclassified patterns is reduced by at least one for the multi-category perceptron cascade network.

![Diagram](image)

**Proof:**
The perceptron cascade is similar to the upstart algorithm except for the fact that each newly added daughter unit is connected to all the previously added daughter units in addition to all the input units. If we set the weights connecting each newly added
daughter to all the previous daughter units to zero, the perceptron cascade would behave exactly as the upstart algorithm. The convergence proof for the perceptron cascade thus follows directly from the proof of the upstart algorithm.

2.6 Tiling Algorithm

The tiling algorithm [Mézard & Nadal, 89] constructs a strictly layered network of threshold neurons. The bottom-most layer of neurons receives inputs from each of the $N$ input neurons. The neurons in each subsequent layer receive inputs from the neurons 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 units 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, 89].

The proposed extension to multiple output classes involves constructing layers with $M$ master neurons (one for each of the output classes). 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.

2.6.1 Multi-Category Tiling Algorithm

1. Train a layer of $M$ master neurons. Each master neuron is connected to the $N$ inputs.

2. If the master neurons of the current layer 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.

   (c) Add a set of $k$ ($1 \leq k \leq M$) ancillary units where $k$ is the number of target classes represented in the set of patterns identified in the above step and train them.
(d) Repeat these last three steps (of adding and training ancillary units) 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.

2.6.2 Convergence Proof

In the tiling algorithm each hidden layer contains $M$ master units plus several ancillary units to achieve a faithful representation of the patterns in the layer. Let $\tau^p = \langle \tau^p_1, \tau^p_2, \ldots, \tau^p_{M+K} \rangle$ (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) units. $\tau^p_i = \pm 1$ for all $i = 1 \ldots (M + K)$.

**Theorem 5:**

Suppose that all classes in layer $L - 1$ are faithful and that the number of errors of the master units ($e_{L-1}$) is non-zero. There exists a weight setting for the master units of the newly added layer ($L$) such that $e_L < e_{L-1}$.
Proof:
Consider a prototype $\tau^p$ for which the master units at layer $L-1$ do not yield the correct output. i.e., $<\tau^p_1, \tau^p_2, \ldots, \tau^p_M> \neq <C^p_1, C^p_2, \ldots, C^p_M>$. The following weight setting for the master unit $j$ $(j = 1 \ldots M)$ in layer $L$ results in correct output for prototype $\tau^p$ at layer $L$. Also, this weight setting ensures that the outputs of all other prototypes, $\tau^q$, for which the master units at layer $L-1$ produce correct outputs (i.e. $<\tau^q_1, \tau^q_2, \ldots, \tau^q_M> = <C^q_1, C^q_2, \ldots, C^q_M>$), are unchanged.

$$ W_{L_j,0} = 2C^p_j $$
$$ W_{L_j,L-1_k} = C^p_j \tau^p_k \text{ for } k = 1 \ldots U_{L-1}, k \neq j $$
$$ W_{L_j,L-1_j} = U_{L-1} $$

For prototype $\tau^p$:

$$ n^p_{L_j} = W_{L_j,0} + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} \tau^p_k $$
$$ = 2C^p_j + U_{L-1} \tau^p_j + (U_{L-1} - 1)C^p_j $$
$$ = U_{L-1} \tau^p_j + (U_{L-1} + 1)C^p_j $$
$$ O^p_{L_j} = \text{sgn}(n^p_{L_j}) $$
$$ = C^p_j $$

For prototype $\tau^q$ (as described above) $\neq \tau^p$:

$$ n^q_{L_j} = W_{L_j,0} + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} \tau^q_k $$
$$ = 2C^p_j + U_{L-1} \tau^q_j + \sum_{k=1,k \neq j}^{U_{L-1}} W_{L_j,L-1_k} \tau^q_k $$
\[ n_{L_j}^q = 2C_j^p + U_{L-1} \tau_j^q + \sum_{k=1, k \neq j}^{U_{L-1}} C_j^p \tau_k^q \tau_k^q \]

**CASE I:**

\( \tau_j^q \neq \tau_j^p \) and \( \tau_k^q = \tau_k^q \) for \( 1 \leq k \leq U_{L-1}, k \neq j \)

Since \( \tau^q \) is correctly classified at layer \( L-1 \) whereas \( \tau^p \) is not, \( \tau_j^q = C_j^p \) since \( \tau_j^q = -\tau_j^p \) and \( C_j^p = -\tau_j^p \).

\[
C_j^p + U_{L-1} \tau_j^q + \sum_{k=1, k \neq j}^{U_{L-1}} C_j^p \tau_k^q \tau_k^q
\]

**CASE II:**

\( \tau_k^q \neq \tau_k^p \) for some \( k, 1 \leq k \leq U_{L-1}, k \neq j \)

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

\[
O_{L_j}^q = sgn(n_{L_j}^q)
\]

Once again we rely on algorithm \( L_W \) to find an appropriate weight setting. With the above weights the previously incorrectly classified prototype, \( \tau^p \), would be corrected and all other prototypes that were correctly classified would be unaffected. This reduces the number of incorrect prototypes by one (i.e. \( \epsilon_L < \epsilon_{L-1} \)). 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.

\[ 3 \quad \text{Experimental Results} \]

This section presents results of some experiments with the multi-category constructive learning algorithms described in section 2.
3.1 Data Sets

The algorithms were tested on two data sets: an artificially generated 3-category data set where 5 bit boolean patterns were assigned randomly to one of three classes (5 such data sets were generated for the experiments); and a real world non linearly separable data set (iris). The original iris data set comprises of 4 real valued attributes and 3 output classes. Since all of the constructive learning algorithms explored in this paper require binary or bipolar representation of input patterns, we had to use a quantized version of the data set comprising of 22 binary (or bipolar as appropriate) valued attributes. (The quantization used ensures that no two patterns belonging to different categories map to the same binary or bipolar input vector). The simulations for tower, pyramid, and tiling algorithms were carried out using bipolar valued patterns while those for the upstart and perceptron cascade algorithms were performed using binary valued patterns. The entire training set of 32 patterns in the case of random mappings and 150 patterns in the case of iris was used for training.

3.2 Training Methodology

For intermediate training of a neuron or a group of neurons the pocket algorithm with ratchet modification was used with the learning rate \( \eta \) set to 1 and the initial weights set randomly to \(-1, 0, \) or 1. In each case, for intermediate training, patterns were randomly drawn from the training set. 64,000 pattern presentations were made for the random mappings and 150,000 pattern presentations were made for iris. These choices were dictated by our choice of the pocket algorithm for training individual threshold neurons (which requires a sufficiently large number of pattern presentations randomly selected from the training set).

In the case of the upstart and perceptron cascade algorithms, several runs failed to converge to zero classification errors. Upon closer scrutiny of the experiments, this was explained by the fact that the training sets of the daughter units had very few patterns with a target output of 1. The pocket algorithm with ratchet modification 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 \( L_W \) in this case. This resulted in the added daughter unit’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 units without converging. To overcome this problem, a balancing of the training set for the daughter unit was performed as follows. If for a daughter unit the fraction of training patterns with target output 1 was less than 25% of the entire training set then the patterns with target output 1 were replicated sufficient number of times so as to create a modified training set with equal number of patterns with targets 1 and 0. Given the tendency of the pocket algorithm to find a set of weights that correctly classify a near-maximal subset of its training set, it was able to now (with the modified training set) at least approximately satisfy the requirements imposed on
The results reported in the following subsection are based on this modification to the training procedure for the upstart and perceptron cascade algorithms.

### 3.3 Results

A summary of the number of units (excluding the input units) generated by each algorithm appears in Tables 1 and 2 for the 5 bit random mappings and the iris data set respectively. The results represent an average of 5 different runs for each data set.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Data Set 3</th>
<th>Data Set 4</th>
<th>Data Set 5</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tower</td>
<td>16.2</td>
<td>19.8</td>
<td>27.6</td>
<td>25.2</td>
<td>23.4</td>
<td>22.44</td>
</tr>
<tr>
<td>Pyramid</td>
<td>17.4</td>
<td>17.4</td>
<td>18</td>
<td>18</td>
<td>22.2</td>
<td>18.6</td>
</tr>
<tr>
<td>Upstart</td>
<td>10.8</td>
<td>10.8</td>
<td>14.6</td>
<td>12.2</td>
<td>13</td>
<td>12.8</td>
</tr>
<tr>
<td>Cascade</td>
<td>11.2</td>
<td>12</td>
<td>12.8</td>
<td>11.2</td>
<td>13.6</td>
<td>12.16</td>
</tr>
<tr>
<td>Tiling</td>
<td>21.8</td>
<td>19.6</td>
<td>21.2</td>
<td>18.6</td>
<td>20.2</td>
<td>20.28</td>
</tr>
</tbody>
</table>

Table 1: Average number of units for 3-class 5-bit random functions

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iris Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tower</td>
<td>6</td>
</tr>
<tr>
<td>Pyramid</td>
<td>6</td>
</tr>
<tr>
<td>Upstart</td>
<td>4</td>
</tr>
<tr>
<td>Cascade</td>
<td>4</td>
</tr>
<tr>
<td>Tiling</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Average number of units for the iris data set

### 4 Summary and Discussion

Constructive neural network learning algorithms offer a potentially powerful approach to inductive learning for pattern classification applications. In this paper, we have focused on a family of such algorithms that incrementally construct networks of threshold neurons with binary or bipolar input patterns. Although a number of such algorithms have been proposed in the literature, most of them were limited to 2-category pattern classification tasks. This paper extends several existing constructive learning algorithms to handle multi-category classification. While the extensions are rather straightforward in the case of some of the algorithms considered, in other cases, they are non-trivial and offer an interesting range of design choices (each with its performance implications) that remain to be explored in detail. However, 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.
In each case, 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 such modification and assuming a weight modification algorithm $L_W$ 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 are currently available can in practice, satisfy the requirements imposed on $L_W$, 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 threshold neuron training algorithms that (with a high probability) satisfy the requirements imposed on $L_W$ and are at least approximately optimal remains an open research problem. Against this background, the primary purpose of the experiments described in section 3 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 data sets - namely, Gallant’s pocket algorithm with ratchet modification. Detailed theoretical and experimental analysis of the performance of single threshold neuron training algorithms is in progress [Yang et al., 95]. We expect such analysis to be useful in suggesting near-optimal designs for threshold neuron training algorithms for each of the constructive learning algorithms.

A few additional comments on the experimental results presented in section 3 are in order. In particular, it must be pointed out that it is premature to draw any conclusions on the relative efficacies of the different algorithms based on the limited set of experiments that were described above. We have not made any attempt to optimize the performance of the proposed algorithms. A number of such improvements suggest themselves. Perhaps the most important one would involve the use of an appropriate single neuron training algorithm that works well with each of the constructive algorithms. An extensive comparison of the different members of this family of algorithms would be meaningful only after we have explored this and related issues in sufficient detail. It is also worth noting that the real valued iris data set is known to be relatively easy to classify with two of the three classes being separable from each other. Quantization simplifies the task further although it does not render pattern set linearly separable. In light of this fact, exploration of quantization algorithms (designed specifically with classification rather than data compression as the objective) are of interest and are currently under study.

Since our primary focus in this paper was on provably convergent multi-category constructive learning algorithms for pattern classification, we have not addressed a number of important issues in the preceding discussion. Each of the constructive algorithms 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 quite useful in guiding the design of better constructive algorithms. Comparative analysis of performance of various constructive algorithms on a broad range of real-world data sets is currently in progress. Generalization ability of this family of constructive learning algorithms also deserves systematic investigation. Extensions of constructive algorithms to work with multi-valued or real-valued inputs are of interest as well. Another potentially useful extension of multi-category constructive algorithms involves the use of winner-take-all groups of threshold neurons instead of independently trained neurons.

Acknowledgements

The authors would like to thank C-H Chen for patiently reading the manuscript and pointing out errors in two of the proofs.

References


