Constructive Neural Network Learning Algorithms for Pattern Classification

Rajesh Parekh
Allstate Research and Planning Center
321 Middlefield Road, Menlo Park CA 94025, USA
rpare@allstate.com

Jihoon Yang
Department of Computer Science, Iowa State University
Ames IA 50011, USA
yang@cs.iastate.edu

Vasant Honavar *
Department of Computer Science, Iowa State University
Ames IA 50011, USA
honavar@cs.iastate.edu
URL: `http://www.cs.iastate.edu/~honavar/aigroup.html`

October 20, 1998

Abstract

Constructive learning algorithms offer an attractive approach for the incremental construction of near-minimal neural network architectures for pattern classification. They help overcome the need for *ad hoc* and often inappropriate choices of network topology in the use of algorithms that search for suitable weights in *a priori* fixed network architectures. Several such algorithms have been proposed in the literature and are shown to converge to zero classification errors (under certain assumptions) on tasks that involve learning a *binary* to *binary* mapping (i.e., classification problems involving binary valued input attributes and two output categories). We present two constructive learning algorithms **MPyramid-real** and **MTiling-real** that extend the *pyramid* [19] and *tiling* [32] algorithms respectively for learning *real* to *M-ary* mappings (i.e., classification problems involving real valued input attributes and multiple output classes). We prove the convergence of these algorithms and empirically demonstrate their applicability on practical pattern classification problems. Additionally, we show how the incorporation of a local pruning step can eliminate several redundant neurons from *MTiling-real* networks.

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

1 Introduction

Artificial neural networks have been successfully applied to solve problems in *pattern classification, function approximation, optimization, pattern matching and associative memories* [10, 20, 31]. Multilayer feedforward networks trained using the *backpropagation* learning algorithm [45] are limited to searching for a suitable set of weights in an *a priori* fixed network topology. This makes it important to select an appropriate network topology for the learning problem on hand. However, there are no known efficient methods for determining the optimal network topology for a given problem. Too small networks are unable to adequately learn the problem well while overly large networks tend to overfit the training data and consequently result in poor generalization performance (see [12] for an analogy to the *curve fitting* problem). In practice, a variety of architectures are tried out and the one that appears best suited to the given problem is picked. Such a *trial-and-error* approach is not only computationally expensive but also does not guarantee that the selected network architecture will be close to optimal or will generalize well. This suggests the need for algorithms that learn both the network topology and the weights.

1.1 Constructive Neural Network Learning Algorithms

Constructive (or *generative*) learning algorithms offer an attractive framework for the incremental construction of near-minimal neural network architectures. These algorithms start with a small network (usually a single neuron) and dynamically grow the network by adding and training neurons as needed until a satisfactory solution is found [13, 15, 19, 23, 29, 32]. Some key motivations [20, 24] for studying constructive neural network learning algorithms are:

- *Flexibility of exploring the space of neural network topologies*
  Constructive algorithms overcome the limitation of searching for a solution in the weight space of an *a priori* fixed network architecture by extending the search, in a controlled fashion, to the entire space of neural network topologies. Further, it has been shown that at least in principle, algorithms that are allowed to add neurons and weights represent a class of *universal learners* [3].

- *Potential for matching the intrinsic complexity of the learning 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, and connectivity) is commensurate with the intrinsic complexity of the underlying learning task (implicitly specified by the training data) [27]. Constructive algorithms search for small solutions first and thus offer a potential for discovering a near-minimal network that suitably matches the complexity of the learning task. Smaller networks are also preferred because of their potential for more efficient hardware implementation and greater transparency in extracting the learned knowledge.

- *Estimation of expected case complexity of the learning task*
  Most practical learning problems are known to be computationally hard to solve. However, little is known about the *expected* case complexity of problems encountered and successfully solved by living systems primarily because it is difficult to mathematically characterize the properties of such problems. Constructive algorithms, if successful, can

provide useful empirical estimates of the expected case complexity of practical learning problems.

- *Trade-offs among performance measures*
  Different constructive learning algorithms allow trading off certain performance measures (e.g., learning time) for others (e.g., network size and generalization accuracy) [48].

- *Incorporation of prior knowledge*
  Constructive algorithms provide a natural framework for incorporating problem-specific knowledge into initial network configurations and for modifying this knowledge using additional training examples [14, 35, 36].

- *Lifelong Learning*
  Recent research in *lifelong learning* [49] has proposed training networks to learn to solve multiple related problems by building on the knowledge acquired from the simpler problems in learning the more difficult ones. Constructive learning algorithms lend themselves well to the lifelong learning framework. A network that has domain knowledge from the simpler task(s) built into its architecture (either by explicitly setting the values of the connection weights or by training them) can form a building block for a system that constructively learns more difficult tasks.

1.2 Network Pruning

Network pruning offers another prominent approach for dynamically determining an appropriate network topology. Pruning techniques (see [41] for an excellent survey) begin by training a larger than necessary network and then eliminate weights and neurons that are deemed redundant. Constructive algorithms offer several significant advantages over pruning based algorithms including the ease of specification of the initial network topology, better economy in terms of training time and number of training examples, and potential for converging to a smaller network with superior generalization [28]. In this paper we will focus primarily on constructive learning algorithms. In section 4 we show how a local pruning pruning step can be integrated into the network construction process to obtain more compact networks.

1.3 Constructive Algorithms for Pattern Classification

Neural network learning can be specified as a *function approximation* problem where the goal is to learn an unknown function $\mathbf{f} : \mathcal{R}^N \longrightarrow \mathcal{R}$ (or a good approximation of it) from a set of input output pairs $S = \{(\mathbf{x}^N, y) \mid \mathbf{x}^N \in \mathcal{R}^N, y \in \mathcal{R}\}$. A variety of constructive neural network learning algorithms have been proposed for solving the general function approximation problem (see [28] for a survey). These algorithms typically use a *greedy strategy* wherein each new neuron added to the network is trained to minimize the residual error as much as possible. Often the unknown target function ($\mathbf{f}$) is inherently complex and cannot be closely approximated by a network comprising of a single hidden layer of neurons implementing simple transfer functions (e.g., *sigmoid*). To overcome this difficulty, some constructive algorithms use the *gaussian* [21] or some more sophisticated transfer function while others such as the *projection pursuit regression* [18] use a summation of several *nonlinear* transfer functions. Alternatively, algorithms such as the *cascade correlation* family construct multilayer networks wherein the structural inter-connections among the hidden neurons allow the network to approximate complex functions using relatively simple neuron transfer functions like the sigmoid [13, 40, 51].

Pattern classification is a special case of function approximation where the function's output y is restricted to one of M ($M \geq 2$) discrete values (or classes) i.e., it involves a *real* to *M-ary* mapping. A neural network for solving classification problems has N input neurons and M output neurons. The k^{th} output neuron ($1 \leq k \leq M$) is trained to output 1 (while all the other output neurons are trained to output 0) for patterns belonging to the k^{th} class[1]. Pattern classification can be described as a *real* to *M-ary* function mapping. Clearly, the class of constructive algorithms discussed above (which implement the more general *real* to *real* mapping) can be adapted for application to pattern classification For example, the use of a cascade correlation type network construction strategy for learning *Bayesian discriminant functions* is described in [55]. However, a special class of constructive learning algorithms can be designed to closely match the unique demands of pattern classification. Since it is sufficient for each output neuron to be binary valued (i.e., output 0 or 1), individual neurons can implement the simple *threshold* or *hard-limiting* activation function (with outputs 0 and 1) instead of a continuous activation function like the sigmoid. Threshold neurons offer the following advantages over their continuous counterparts: Firstly, they are potentially easier to implement in hardware. Secondly, the *perceptron learning rule* [43] is a simple iterative procedure for training threshold neurons. The learning rules for sigmoid neurons and the like are more complicated and thus computationally more expensive. Thirdly, threshold functions can be clearly described in terms of simple *"if-then-else"* rules. This makes it easier to incorporate domain expertise (which is usually available in the form of if-then-else rules) into a network of threshold neurons [14][2]. Similar argument suggests that the task of extracting learned knowledge from a network of threshold neurons would be considerably simpler. In this paper, we will focus on constructive learning of networks of threshold neurons for pattern classification.

1.3.1 Constructive Learning using Iterative Weight Update

A number of algorithms that incrementally construct networks of threshold neurons for learning the *binary* to *binary* mapping have been proposed in the literature (for example, the *tower*, *pyramid* [19], *tiling* [32], *upstart* [15], *oil-spot* [30], and *sequential* [29] algorithms). These algorithms differ in terms of their choices regarding: restrictions on input representation (e.g., binary or bipolar 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. They can be shown to converge to networks which yield zero classification errors on any non-contradictory training set involving two output classes (see [48] for a unifying framework that explains the convergence properties of these constructive algorithms). A gemoetrical analysis of the decision boundaries of some of these algorithms is presented in [7]. Practical pattern classification often requires assigning patterns to M (where $M > 2$) categories. Although in principle, the M category classification task can be decomposed into M 2-category classification tasks, this approach does not take

[1] A single output neuron suffices in the case of problems that involve two category classification.

[2] Note that even the KBANN algorithm [50] that uses the backpropagation algorithm for connectionist theory refinement treats the individual sigmoid neurons as threshold neurons for the purpose of incorporating the domain theory into an initial neural network.

into account the inter-relationships between the M output categories. Further, each of the constructive algorithms mentioned above have been designed to operate on binary (or bipolar) valued attributes only. Real-valued attributes are almost invariably encountered in practical classification tasks. One work around for this problem is to discretize the real-valued attributes prior to training. Although a variety of discretization techniques have been proposed in the literature (see [11] for a survey), these can result in a loss of information and also vastly increase the number of input attributes. Thus, it is of interest to design algorithms that can directly accept real-valued attributes. The *perceptron cascade* learning algorithm [6] uses a projection based idea for handling real valued attributes in two category pattern classification. We present constructive neural network learning algorithms that are capable of handling multiple output categories and real-valued pattern attributes.

1.3.2 Exploiting Geometric Properties for Constructive Learning

The class of constructive learning algorithms we focus on in this paper trains individual neurons using an iterative weight update strategy (such as the *perceptron* rule). Another class of constructive learning algorithms that use a *one-shot* learning strategy deserves mention. These algorithms exploit the geometric properties of the training patterns to directly (i.e., in one-shot) determine appropriate weights for the neurons added to the network. The *Grow and Learn* (GAL) algorithm [1] and the *DistAl* algorithm [54] construct a single hidden layer network that implements a kind of *nearest neighbor classification* scheme. Each hidden neuron is an *exemplar* representing a group of patterns that belong to the same class and are close to each other in terms of some suitably chosen distance metric. The *minimizing resources* method [44], the *multisurface* method [4], and the *voronoi diagram* approach [5] are based on the idea of *partitioning* the input space by constructing linear hyperplanes. The partition identifies regions within the input space where each region represents patterns belonging to one particular output class. The hidden layer neurons implement hyperplanes that identify these regions and the output layer neurons combine regions that represent the same output class. The geometric approach to constructive learning can be applied successfully in solving small to medium scale problems. However, the global search strategy employed by these algorithms can pose a limitation when learning from very large training sets [48]. Further, the reliance on a suitably chosen distance metric (in the case of *GAL* and *DistAl*) makes it imperative for the user to try out a variety of distance metrics for each learning problem.

In this paper, we present extensions of the *pyramid* and the *tiling* algorithms to handle multiple output classes and real-valued pattern attributes[3]. We prove the convergence of these algorithms and demonstrate their applicability on some practical problems. The remainder of this paper is organized as follows: Section 2 gives an overview of some elementary concepts and describes the notation used throughout this paper. Sections 3 and 4 describe the *MPyramid-real* and *MTiling-real* constructive learning algorithms respectively and prove their convergence. Section 5 illustrates the practical applicability of these algorithms and section 6 concludes with a discussion and some directions for future research.

[3] The framework presented here is more general and can potentially be applied to the entire class of constructive algorithms for pattern classification. The interested reader is referred to [37] for an application of this framework to the *tower*, *upstart*, *perceptron cascade*, and *sequential* learning algorithms.

2 Preliminaries

2.1 Threshold Logic Units

A N-input *threshold logic unit* (TLU, also known as a *perceptron*) is an elementary processing unit that computes the threshold (hard-limiting) function of the weighted sum of its inputs. The output (O^p) of a TLU with weights $\mathbf{W} =< W_0, W_1, W_2, \ldots, W_N >$ (where the weight W_0 is referred to as the *threshold* or *bias*) in response to a pattern $\mathbf{X}^p =< X_1^p, X_2^p, \ldots, X_N^p >$ is $O^p = 1$ if $W_0 + \sum_{i=1}^{N} W_i \cdot X_i \geq 0$ and $O^p = -1$ otherwise [4]. For notational convenience, we prefix each pattern $\mathbf{X}^p$ with a 1 (i.e., $\mathbf{X}^p =< 1, X_1^p, X_2^p, \ldots, X_N^p >$) and denote the TLU output O^p as the hard-limiting function of $\mathbf{W} \cdot \mathbf{X}^p$.

2.1.1 Perceptron Learning Rule

A N-input TLU implements a $(N-1)$-dimensional hyperplane that partitions the N-dimensional Euclidean pattern space defined by the coordinates $X_1, \ldots, X_N$ into two regions (or classes). A TLU can thus function as a 2-category classifier. Consider 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 for the input pattern $\mathbf{X}^p$ and p ranges from 1 to $|S|$). A TLU can be trained using the *perceptron learning rule* [43] ($\mathbf{W} \leftarrow \mathbf{W} + \eta(C^p - O^p)\mathbf{X}^p$ where $\eta > 0$ is the learning rate) 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 $\forall \mathbf{X}^q \in S^-$, $\hat{\mathbf{W}} \cdot \mathbf{X}^q < 0$. If such a weight vector ($\hat{\mathbf{W}}$) exists then the pattern set S is said to be *linearly separable*.

2.1.2 Stable Variants of the Perceptron Rule

If the set S is not linearly separable then the perceptron algorithm behaves poorly in the sense that the classification accuracy on the training set can fluctuate widely from one training epoch to the next. Several modifications to the perceptron algorithm (e.g., the *pocket algorithm* with *ratchet modification* [19], the *thermal perceptron algorithm* [16], the *loss minimization algorithm* [25], and the *barycentric correction procedure* [39]) are proposed 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. Siu *et al* have established the necessary and sufficient conditions for a training set S to be non-linearly separable [47]. They have also shown that the problem of identifying a largest linearly separable subset of S is NP-complete. Thus, we rely on a suitable heuristic algorithm (such as the *pocket algorithm* with *ratchet modification*) to correctly classify as large a subset of training patterns as possible in the limited training time allowed. We denote such an algorithm by $\mathcal{A}$. In our experiments with constructive learning algorithms we use the *thermal perceptron algorithm* to train individual TLUs. The weight update equation of the *thermal perceptron algorithm* is:

$$\mathbf{W} \leftarrow \mathbf{W} + \eta \frac{1}{T}(D^p - O^p)\mathbf{X}^p e^{-|n^p|/T}$$

where n^p is the net input ($\mathbf{W} \cdot \mathbf{X}^p$) and T is the temperature. T is set to an initial value T_0 at the start of learning and gradually annealed to 0 as the training progresses. The damping factor

[4] Although *bipolar* TLUs whose outputs are 1 and -1 is functionally equivalent to *binary* TLUs whose outputs are 1 and 0 empirical evidence suggests that networks constructed using bipolar TLUs are often smaller than networks constructed using binary neurons for the same task [20].

$(e^{-|n^p|/T})$ prevents any large weight changes towards the end of the training thereby avoiding any irreversible deterioration in the TLU's classification accuracy.

2.2 Multiclass Discrimination

Classification problems involving M ($M > 2$) output classes require a layer of M TLUs. These TLUs can be trained by *independent* training or as a *winner-take-all* (WTA) group. The former strategy trains the TLUs independently and in parallel. However, this does not take into account the fact that class assignments are crisp (i.e., pattern assigned to class i cannot possibly belong to any other class as well) and thus potentially results in scenarios where more than one TLU has an output of 1. The WTA training strategy gears the weight changes so that the i^{th} TLU has the highest net input among the group of M TLUs in response to a pattern belonging to class i. The winner (i.e., the neuron with the highest net input) is assigned an output of 1 while all other neurons are assigned outputs of -1. In the event of a tie for the highest net input all neurons are assigned outputs of -1. If a pattern is misclassified then the weights of the TLUs whose output in response to the pattern does not match the desired output are updated using the perceptron rule (or one of its variants). WTA training offers an advantage over independent training in that pattern classes that are only pairwise separable from each other can be correctly classified using WTA training while in independent training only pattern classes that are independently separable from all the other classes can be correctly classified [20].

2.3 Preprocessing

Most constructive learning algorithms are designed for binary (or bipolar) valued inputs. An extension of the *upstart* algorithm [46] and the *perceptron cascade* algorithm [6] proposed a preprocessing technique to handle patterns with real valued attributes wherein the patterns are projected on to a parabolic surface by appending to each pattern ($\mathbf{X}^p =< X_1^p, \ldots, X_N^p >$) an additional attribute $X_{N+1}^p = \sum_{i=1}^{N}(X_i^p)^2$. With this projection it is possible to train a TLU to exclude any one pattern from all others such that the TLU outputs 1 for the pattern to be excluded and -1 for all the others. We use this projection idea to demonstrate the convergence of the *MPyramid-real* algorithm on real-valued pattern attributes (see section 3.1). Even the *tiling* algorithm was designed to work only with bipolar valued patterns. However, as we show in the formal convergence proof (section 4.1), it is not necessary to preprocess the patterns with real-valued attributes to guarantee convergence of the *MTiling-real* algorithm. However, preprocessing the patterns as described above will not hamper the convergence properties of the algorithm.

2.4 Notation

The following notation is used in the description of the algorithms and their convergence proofs:
Number of inputs: N
Number of outputs: M
Input layer index: I
Indices for other layers (hidden and output): 1, 2, ..., L
Number of neurons in layer l: $|l|$

Indexing of neurons in layer l: $l_1, l_2, \ldots, l_{|l|}$
Weight vector of neuron i in layer l: $\mathbf{W}_{l_i} =< W_{l_i,0}, W_{l_i,1}, \ldots, W_{l_i,|l|} >$, $W_{l_i,k} \in \mathcal{R}$, $k = 0 \ldots |l|$
Connection weight between neuron i in layer l^1 and neuron j in layer l^2: $W_{l^1_i, l^2_j}$
Pattern set: $S = \{\mathbf{X}^1, \mathbf{X}^2, \ldots, \}$
Pattern p: $\mathbf{X}^p = < X_1^p, \ldots, X_N^p >$ where $X_i^p \in \mathcal{R}$ for all i and $1 \leq p \leq |S|$
Augmented pattern p: $\mathbf{X}^p = < 1, X_1^p, \ldots, X_N^p >$
Projected pattern p: $\hat{\mathbf{X}}^p = < 1, X_1^p, \ldots, X_N^p, X_{N+1}^p >$, $X_{N+1}^p = \sum_{i=1}^N (X_i^p)^2$
Net input of neuron l_j in response to pattern $\mathbf{X}^p$: $n_{l_j}^p$
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 $\mathbf{X}^p \in$ class i and $C_i^p = -1$ otherwise
Layer l's output in response to the pattern $\mathbf{X}^p$: $\mathbf{O}_l^p = < O_{l_1}^p, O_{l_2}^p, \ldots, O_{l_{|l|}}^p > O_{l_i}^p \in \{-1, +1\}$ for all i
Number of patterns incorrectly classified by layer l[5]: e_l

Define a function $sgn : \mathcal{R} \longrightarrow \{-1, 1\}$ as $sgn(x) = -1$ if $x < 0$ and $sgn(x) = 1$ if $x \geq 0$. Note that bipolar TLUs implement the *sgn* function of their net input. The input layer neurons are designed to allow the patterns to be input to the network and thus simply copy their input to their output.

3 The *MPyramid-real* Algorithm

The *pyramid* algorithm [19] constructs a layered network of TLUs by successively placing each new TLU above the existing ones. The first neuron receives inputs from the N input neurons. Each succeeding neuron receives inputs from the N input neurons and from each of the neurons below itself. Thus, the second neuron receives a total of $N + 1$ inputs, the third neuron receives a total of $N + 2$ inputs and so on. Each newly added neuron takes over the role of the output neuron. The network growth continues until the desired classification accuracy is achieved.

The extension of the pyramid algorithm to handle real-valued attributes involves modifying each input pattern by augmenting an extra attribute (X_{N+1}^p) as described in section 2.3. The network thus has $N + 1$ input neurons. To handle multiple output categories the algorithm uses M neurons (instead of one) in each layer of the network. The newly added layer of M neurons becomes the network output layer. Each of the M neurons in the new layer are connected to the $N + 1$ input neurons and to each of the M neurons in each preceding layer that was added to the network. This algorithm is described in Fig. 1 and the resulting network is shown in Fig. 2.

3.1 Convergence Proof

Theorem 1 *There exists a set of weights for neurons in the newly added layer L of the network such that the number of misclassifications is reduced by at least one (i.e., $\forall L > 1,\ e_L < e_{L-1}$).*

Proof:
Define $\kappa = \max\limits_{p,q} \sum\limits_{i=1}^{N} (X_i^p - X_i^q)^2$. For each pattern $\hat{\mathbf{X}}^p$, define $\epsilon_p = \frac{1}{2} \min\limits_{q \neq p} \sum\limits_{i=1}^{N} (X_i^p - X_i^q)^2$. It is clear that $0 < \epsilon_p < \kappa$ for all patterns $\hat{\mathbf{X}}^p$. Assume that a pattern $\hat{\mathbf{X}}^p$ is not correctly classified

[5] A pattern is said to be correctly classified at layer l when $\mathbf{C}^p = \mathbf{O}_l^p$.

```
Algorithm: MPyramid-real

Input:    A training set $S$
Output: A trained MPyramid-real network

begin
1)  Set the current output layer index $L = 0$
2)  repeat
        // Construct a new output layer and train it
        a. $L = L + 1$
        b. Add $M$ output neurons to the network at layer $L$
        c. Connect each newly added neuron to all the input neurons and
           to each neuron in each of the preceding layers, if there exist any
        d. Train the weights of the newly added neurons using the algorithm $\mathcal{A}$
           (Note that all other weights of the network remain frozen)
    until ($current\_accuracy \geq DESIRED\_ACCURACY$ or $L \geq MAX\_LAYERS$)
end
```

Figure 1: *MPyramid-real* Algorithm.

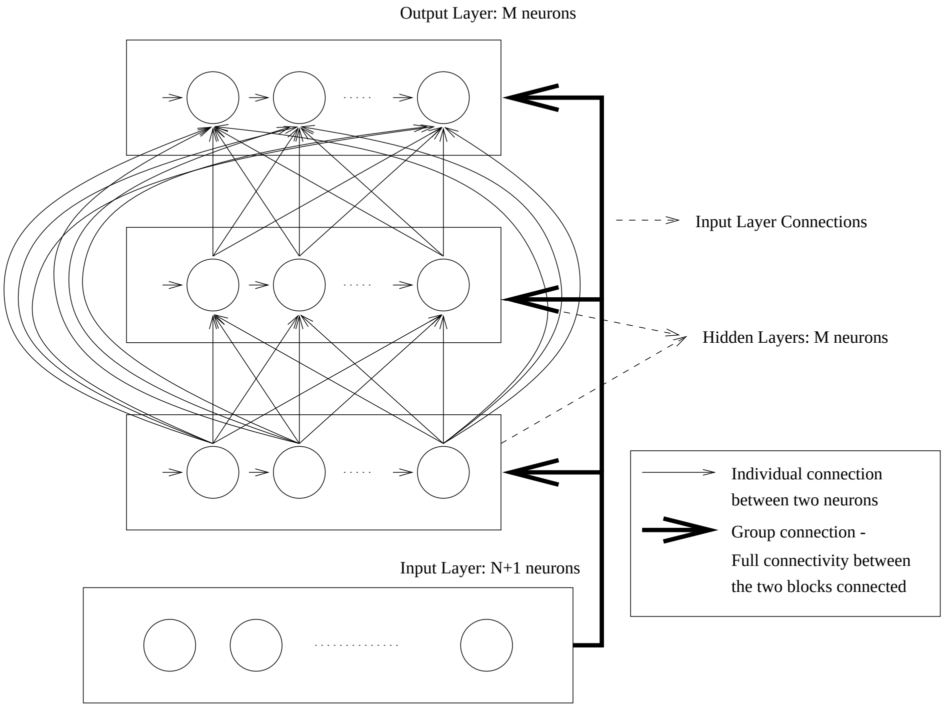

Figure 2: *MPyramid-real* Network.

at layer $L-1$ (i.e., $\mathbf{C}^p \neq \mathbf{O}^p_{L-1}$). Further, let the output vector $\mathbf{O}^p_{L-1}$ for the misclassified pattern $\hat{\mathbf{X}}^p$ be such that $O^p_{L-1_\beta} = 1$ and $O^p_{L-1_k} = -1$, $\forall k = 1 \ldots M, k \neq \beta$; whereas the target output $\mathbf{C}^p$ is such that $C^p_\gamma = 1$ and $C^p_l = -1$, $\forall l = 1 \ldots M, l \neq \gamma$, and $\gamma \neq \beta$.

The network adds a new layer L of M neurons. Let the weights of these new neurons L_j $(j = 1 \ldots M)$ be set as follows (see Fig. 3):

$$\begin{aligned}
W_{L_j,0} &= C^p_j(\kappa + \epsilon_p - \sum_{i=1}^{N}(X^p_i)^2) \\
W_{L_j,I_i} &= 2C^p_j X^p_i \text{ for } i = 1 \ldots N \\
W_{L_j,I_{N+1}} &= -C^p_j \\
W_{L_j,L-k_i} &= 0 \text{ for } k = 2 \ldots L-1, \text{ and } i = 1 \ldots M \\
W_{L_j,L-1_j} &= \kappa \\
W_{L_j,L-1_i} &= 0 \text{ for } i = 1 \ldots M, i \neq j
\end{aligned} \tag{1}$$

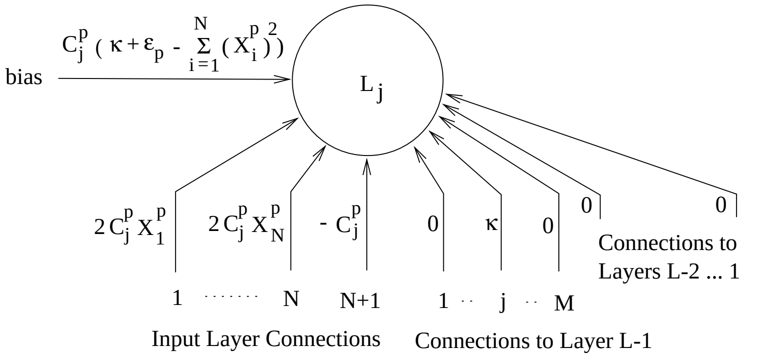

Figure 3: Weight Setting for the Output Neuron L_j of the *MPyramid-real* Network.

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

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

Thus, the net inputs of the output neurons L_γ, L_β, and L_j where $j = 1 \ldots M; j \neq \gamma, j \neq \beta$ are:

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

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

Since $\epsilon_p > 0$, for all p, the net input of neuron L_γ is higher than that of every other neuron in the layer L. Thus, $O^p_{L_\gamma} = 1$ and $O^p_{L_j} = -1$, $\forall j \neq \gamma$ which means that pattern $\hat{\mathbf{X}}^p$ is correctly classified at layer L. Even if as a result of a tie for the highest net input, the output of each neuron in layer $L-1$ in response to $\hat{\mathbf{X}}^p$ is $O^p_{L-1_j} = -1$ the weights of the new neurons in layer L would result in a correct classification of $\hat{\mathbf{X}}^p$.

Consider the pattern $\hat{\mathbf{X}}^q \neq \hat{\mathbf{X}}^p$ that is correctly classified at layer $L-1$ (i.e., $\mathbf{O}^q_{L-1} = \mathbf{C}^q$).

$$
\begin{aligned}
n^q_{L_j} &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^q_i + \sum_{k=1}^{L-1}\sum_{i=1}^{M} W_{L_j,L-k_i} O^q_{L-k_i} \\
&= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X^q_i + \sum_{i=1}^{M} W_{L_j,L-1_i} O^q_{L-1_i} \\
&\quad \text{since } W_{L_j,L-k_i} = 0 \text{ for } k = 2,\ldots,L-1, \text{ and } i = 1,\ldots,M \text{ (see equation 1)} \\
&= C^p_j(\kappa + \epsilon_p - \sum_{i=1}^{N}(X^p_i)^2) + 2C^p_j \sum_{i=1}^{N}(X^p_i)(X^q_i) - C^p_j \sum_{i=1}^{N}(X^q_i)^2 + \kappa O^q_{L-1_j} \\
&= C^p_j(\kappa + \epsilon_p) + \kappa O^q_{L-1_j} - C^p_j \sum_{i=1}^{N}[(X^p_i)^2 - 2(X^p_i)(X^q_i) + (X^q_i)^2] \\
&= C^p_j(\kappa + \epsilon_p) + \kappa O^q_{L-1_j} - C^p_j\,[\sum_{i=1}^{N}(X^p_i - X^q_i)^2] \\
&= C^p_j(\kappa + \epsilon_p - \epsilon') + \kappa O^q_{L-1_j} \text{ where } \epsilon' = \sum_{i=1}^{N}(X^p_i - X^q_i)^2; \text{ note } \epsilon' > \epsilon_p \\
&= \kappa' C^p_j + \kappa O^q_{L-1_j} \text{ where } \kappa + \epsilon_p - \epsilon' = \kappa', \text{i.e., } \kappa' < \kappa
\end{aligned}
\qquad (3)
$$

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. Thus, $\mathbf{O}^q_L = \mathbf{O}^q_{L-1} = \mathbf{C}^q$ i.e., the classification of previously correctly classified patterns remains unchanged. We have shown the existence of weights that will reduce the number of misclassifications whenever a new layer is added to the network. We rely on the TLU weight training algorithm $\mathcal{A}$ to find such weights. Since the training set is finite in size, eventual convergence to zero errors is guaranteed. □

3.2 Example

The following example illustrates the concepts described in the above proof. Consider a simple dataset shown in Fig. 4. The patterns belong to 3 output classes and are clearly not linearly separable. Table 1 summarizes the dataset.

By definition, $\kappa = \max_{p,q} \sum_{i=1}^{N}(X^p_i - X^q_i)^2$. For the example dataset, $\kappa = 2$. The first layer of the network (let us designate it by L^1) is trained using the algorithm $\mathcal{A}$. One possible set of

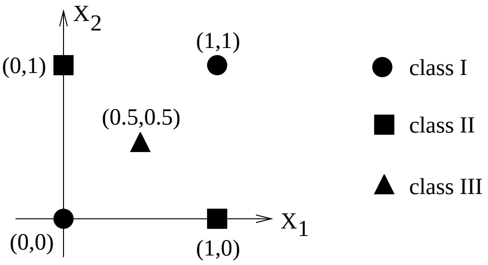

Figure 4: Example Dataset to Illustrate the Convergence Proofs

Table 1: Description of the Example Dataset

Number	Input			Output			ϵ_p
	X_1	X_2	$X_3 = \sum_{i=1}^{2} x_i^2$	c_1	c_2	c_3	
1	0	0	0	-1	-1	1	0.25
2	0	1	1	-1	1	-1	0.25
3	1	0	1	-1	1	-1	0.25
4	1	1	2	-1	-1	1	0.25
5	0.5	0.5	0.5	1	-1	-1	0.25

weights for the neurons is depicted in Fig. 5. The response of each of neurons to the patterns in the dataset is summarized in Table 2.

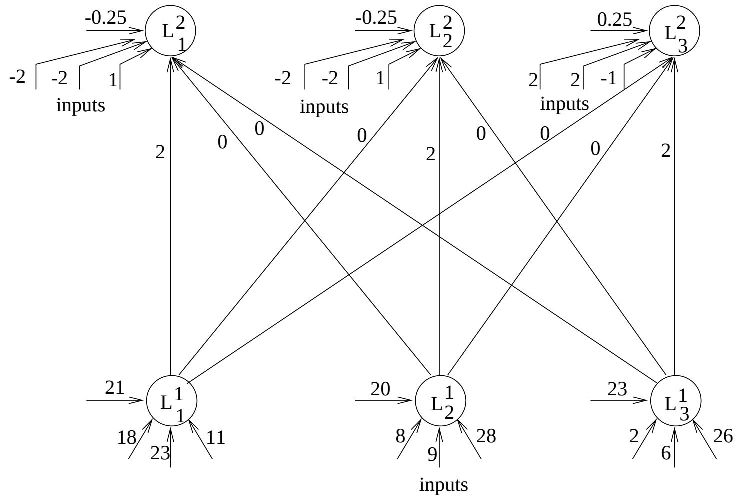

Figure 5: *MPyramid-real* Network for the Example Dataset

The pattern $\hat{\mathbf{X}}^4 =< 1, 1, 2 >$ is misclassified at layer L^1. Let $\hat{\mathbf{X}}^4$ represent the pattern $\hat{\mathbf{X}}^p$ in the proof. The algorithm adds a new layer of neurons (L^2) to the network. Since for $\hat{\mathbf{X}}^4$ the neuron L^1_2 output 1 whereas the neuron L^1_3 should have output 1, L^2_3, L^2_2, and L^2_1 correspond to the neurons L_γ, L_β, and L_j respectively in the proof. Equation 1 specifies one particular set of

Table 2: Response of the Layer L^1 Neurons

Number	Net Input			Output		
	$n_{L^1_1}$	$n_{L^1_2}$	$n_{L^1_3}$	$o_{L^1_1}$	$o_{L^1_2}$	$o_{L^1_3}$
1	21	20	23	-1	-1	1
2	55	57	55	-1	1	-1
3	50	56	51	-1	1	-1
4	84	93	83	-1	1	-1
5	47	42.5	40	1	-1	-1

weights for the newly added neurons (as shown in Fig. 5). The response of the neurons in L^2 to each pattern is given in Table 3. We see the the net inputs of the neurons L_γ, L_β, and L_j in response to pattern $\hat{\mathbf{X}}^4$ are ϵ_p, $-\epsilon_p$ and $-2\kappa - \epsilon_p$ respectively as derived in the proof. Further, the classification of all previously correctly classified patterns such as $\hat{\mathbf{X}}^1$, $\hat{\mathbf{X}}^2$, $\hat{\mathbf{X}}^3$, and $\hat{\mathbf{X}}^5$ that represent the pattern $\hat{\mathbf{X}}^q$ in the proof remains unaltered.

Table 3: Response of the Layer L^2 Neurons

Number	Net Input			Output		
	$n_{L^2_1}$	$n_{L^2_2}$	$n_{L^2_3}$	$o_{L^2_1}$	$o_{L^2_2}$	$o_{L^2_3}$
1	-2.25	-2.25	2.25	-1	-1	1
2	-3.25	0.75	-0.75	-1	1	-1
3	-3.25	0.75	-0.75	-1	1	-1
4	-4.25	-0.25	0.25	-1	-1	1
5	0.25	-3.75	-0.25	1	-1	-1

4 The *MTiling-real* Algorithm

The *tiling* algorithm [32] 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* and a set (possibly empty) of ancillary neurons that are added and trained to ensure a *faithful representation* of the training patterns. The *faithfulness* criterion states that no two training examples belonging to different classes should produce identical output at any given layer. Faithfulness is clearly a necessary condition for convergence in strictly layered networks [32].

The proposed extension to multiple output classes involves constructing layers with M master neurons (one for each of the output classes)[6]. Unlike the *MPyramid-real* algorithm, it is not necessary to preprocess the dataset using projection. Groups of one or more ancillary neurons are trained at a time in an attempt to make the current layer faithful. The algorithm is described in Fig. 6 and the resulting network is shown in Fig. 7.

[6] An earlier version of this algorithm appeared in [53].

```
Algorithm MTiling-real

Input:   A training set $S$
Output: A trained tiling network

begin
  1) Train a single layer network with $M$ output neurons using the algorithm $\mathcal{A}$
     (Note that these $M$ neurons are designated as the master neurons)
  2) Let $L = 1$ denote the number of layers in the network
  3) while ($current\_accuracy < DESIRED\_ACCURACY$ and $L < MAX\_LAYERS$) do
       a. while (layer $L$ is not faithful) do
            // Make the current layer faithful
            Let $O_L$ be the set of outputs of layer $L$ for the patterns in $S$
            For each $v \in O_L$ let $S_v \subseteq S$ be the set of patterns that produced output $v$
               and let $C_v$ be the number of output classes to which the patterns in $S_v$ belong
            // If $C_v > 1$ then the output vector $v$ is unfaithful
            Randomly pick a $v$ for which $C_v > 1$
            Add $C_v$ ancillary neurons to the layer $L$
            Train the ancillary neurons using the algorithm $\mathcal{A}$ to separate the patterns in $S_v$
          end while
       b. $L = L + 1$
       c. Add $M$ master neurons to the new output layer $L$
       d. Connect the neurons in layer $L$ to all neurons in layer $L - 1$
       e. Train the layer $L$ on the patterns of $S$ using the algorithm $\mathcal{A}$
     end while
end
```

Figure 6: *MTiling-real* Algorithm.

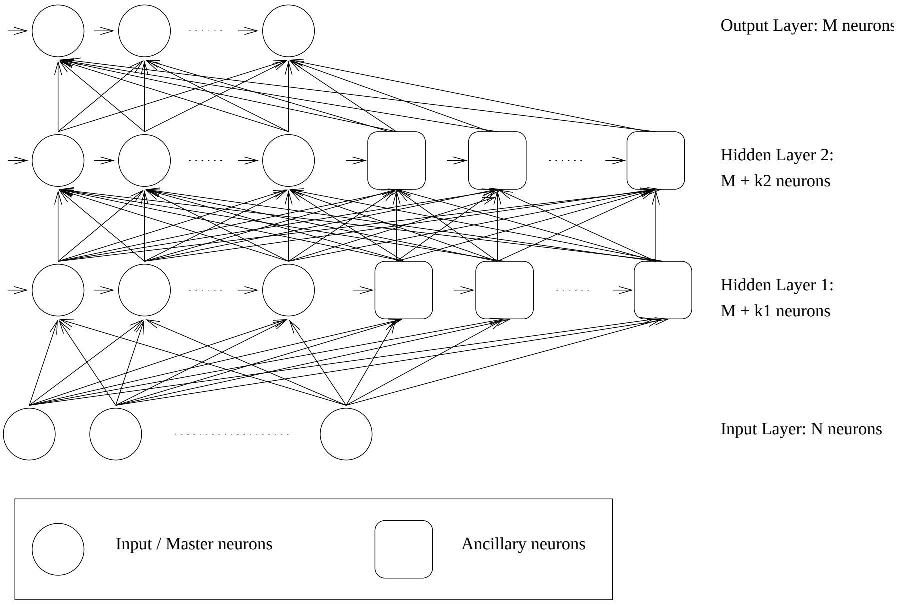

Figure 7: *MTiling-real* Network.

4.1 Convergence Proof

Each hidden layer contains M master neurons and K ($K \geq 0$) ancillary neurons that are trained to achieve a faithful representation of the patterns. Let $\overline{S}$ be a subset of the training set S such that for each pattern $\mathbf{X}^p$ belonging to $\overline{S}$ the outputs $O_1^p, O_2^p, \ldots, O_{M+K}^p$ are exactly the same. We designate this output vector $< O_1^p, O_2^p, \ldots, O_{M+K}^p >$ as a prototype $\mathbf{\Pi}^p =< \pi_1^p, \pi_2^p, \ldots, \pi_{M+K}^p >$, $\pi_i^p = \pm 1$ for all $i = 1 \ldots (M + K)$. If all the patterns of $\overline{S}$ belong to exactly one class (i.e., they have the same desired output) then the prototype $\mathbf{\Pi}^p$ is a faithful representation of the patterns in $\overline{S}$. Otherwise, $\mathbf{\Pi}^p$ is an unfaithful representation of $\overline{S}$. Further, if $< \pi_1^p, \pi_2^p, \ldots, \pi_M^p > = < C_1^p, C_2^p, \ldots, C_M^p >$ (i.e., the observed output for the patterns is the same as the desired output) then the patterns in $\overline{S}$ are said to be correctly classified.

The algorithm's convergence is proved in two parts: first we show that it is possible to obtain a faithful representation of the training set (with real-valued attributes) at the first layer (L^1). Then we prove that with each additional layer the number of classification errors is reduced by at least one.

Theorem 2 *For any finite non-contradictory dataset it is possible to train a layer of threshold neurons such that the output of these neurons is a faithful representation of the dataset.*

Proof:
Assume that a layer (L^1) with M master neurons is trained on the dataset (S). Consider a subset $\overline{S}$ of S such that the master neurons give the same output for each pattern in $\overline{S}$. Further assume that $\overline{S}$ is not faithfully represented by the master neurons. We demonstrate that it is possible to add ancillary neurons (with appropriately set weights) to the layer L^1 in order to obtain a faithful representation of $\overline{S}$.

Consider a pattern $\mathbf{X}^p$ belonging to the *covex hull* [7] of $\overline{S}$. If $\mathbf{X}^p$ is such that some attribute $i \; (i = 1, \ldots, N) \; |X_i^p| > |X_i^q|$ for all $\mathbf{X}^q \in \overline{S}$ and $\mathbf{X}^q \neq \mathbf{X}^p$ then an ancillary neuron with weights $\mathbf{W} = \{-(X_i^p)^2, 0, \ldots, 0, X_i^p, 0, \ldots, 0\}$ (i.e., all weights except W_0 and W_i set to 0) will output 1 for $\mathbf{X}^p$ and -1 for all other patterns. If however, $\overline{S}$ is such that there is a tie for the highest value of each attribute X_i among the patterns then there must exist a pattern $\mathbf{X}^p$ on the convex hull of $\overline{S}$ that dominates all others in the sense that for each attribute X_i, $X_i^p \geq X_i^q$ for all $\mathbf{X}^q$ in $\overline{S}$ (note that $\mathbf{X}^q \neq \mathbf{X}^p$). Clearly, $\mathbf{X}^p \cdot \mathbf{X}^p > \mathbf{X}^p \cdot \mathbf{X}^q$. An ancillary neuron with weights $\mathbf{W} = \{-\sum_{l=1}^{N}(X_l^p)^2, X_1^p, \ldots X_N^p\}$ will output 1 for $\mathbf{X}^p$ and -1 for all other patterns in $\overline{S}$.

After adding an ancillary neuron, the output of the layer L^1 is faithful in response to $\mathbf{X}^p$. Note that this output is distinct from the outputs for all the other patterns in the entire training set S. In effect, the pattern $\mathbf{X}^p$ has been *excluded* from the remaining patterns in the training set. Similarly, using additional TLUs (up to $|\overline{S}|$ TLUs in all) it can be shown that the outputs of the neurons in the layer provide a faithful representation of $\overline{S}$. □

In practice, by training a groups of one or more ancillary neurons at a time 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 total number of training patterns.

Theorem 3 *There exists a set of weights for the master neurons of a newly added layer L in the network such that the number misclassifications is reduced by at least one (i.e., $\forall L > 1, \; e_L < e_{L-1}$).*

Proof:
Consider a set $S_1 \subseteq S$ of patterns that belong to the same output class but are not correctly classified by the master neurons in layer $L - 1$. Let the prototype $\mathbf{\Pi}^p = < \pi_1^p, \pi_2^p, \ldots, \pi_{M+K}^p >$ represent the output of layer $L - 1$ in response to the patterns in S_1. Since the patterns in S_1 are not correctly classified at layer L^1, $< \pi_1^p, \pi_2^p, \ldots, \pi_M^p > \neq < C_1^p, C_2^p, \ldots, C_M^p >$ (the desired output for the patterns in S_1). For the incorrectly classified prototype $\mathbf{\Pi}^p$ assume that $\pi_\beta^p = 1, 1 \leq \beta \leq M$ and $\forall j = 1 \ldots M, j \neq \beta$, $\pi_j^p = -1$. Clearly, $C_\beta^p = -1$ and $\exists \gamma \; 1 \leq \gamma \leq M, \gamma \neq \beta$ such that $C_\gamma^p = 1$.

The algorithm adds a new layer (L) of M master neurons. The following weights for the master neuron L_j ($j = 1 \ldots M$) shown in Fig. 8 results in the correct classification of patterns in S_1. It also ensures that the output of any other set of patterns $S_2 \subseteq S$ (with $S_1 \cap S_2 = \phi$ and the corresponding prototype prototype $\mathbf{\Pi}^q = < \pi_1^q, \pi_2^q, \ldots, \pi_{M+K}^q >$) for which the master neurons of layer $L - 1$ produce correct outputs (i.e., $< \pi_1^q, \pi_2^q, \ldots, \pi_M^q > = < C_1^q, C_2^q, \ldots, C_M^q >$) remains unchanged.

$$\begin{aligned} W_{L_j,0} &= 2C_j^p \\ W_{L_j,L-1_k} &= C_j^p \pi_k^p \text{ for } k = 1 \ldots |L-1|, k \neq j \\ W_{L_j,L-1_j} &= |L-1| \end{aligned} \tag{4}$$

[7] 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. The interested reader is referred to [8] for a detailed description of convex hulls and related topics in computational geometry.

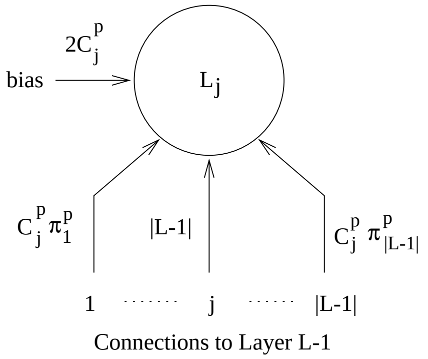

Figure 8: Weight Setting for the Output Neuron L_j of the *MTiling-real* Network.

From equation 4, the net input of neuron L_j in response to the prototype $\mathbf{\Pi}^p$ is:

$$\begin{aligned} n^p_{L_j} &= W_{L_j,0} + \sum_{k=1}^{|L-1|} W_{L_j,L-1_k}\pi^p_k \\ &= 2C^p_j + |L-1|\pi^p_j + \sum_{k=1,k\neq j}^{|L-1|} C^p_j\pi^p_k\pi^p_k \\ &= 2C^p_j + |L-1|\pi^p_j + (|L-1|-1)C^p_j \\ &= |L-1|\pi^p_j + (|L-1|+1)C^p_j \end{aligned} \tag{5}$$

Thus,

$$\begin{aligned} n^p_{L_\gamma} &= |L-1|(-1) + (|L-1|+1)(1) \\ &= 1 \\ n^p_{L_\beta} &= |L-1|(1) + (|L-1|+1)(-1) \\ &= -1 \text{ where } 1 \leq \beta \leq M, \beta \neq \gamma \\ n^p_{L_k} &= |L-1|(-1) + (|L-1|+1)(-1) \text{ for } k = 1\ldots M, k\neq\gamma, k\neq\beta \\ &= -2|L-1|-1 \end{aligned}$$

The master neuron L_γ has the highest net input among all master neurons in layer L which means that $O^p_{L_\gamma} = 1$ and $O^p_{L_j} = -1, \forall j = 1\ldots M, j \neq \gamma$ and $\mathbf{C}^p = \mathbf{O}^p_L$. Thus, the patterns in S_1 are now correctly classified. Even if as a result of a tie in the value of the highest net input among the master neurons of layer $L-1$, $\mathbf{\Pi}^p$ is such that $\pi^p_i = -1$ for all $i = 1\ldots M$, we see from equation 5 that the net input of neuron L_γ is still the largest among the net inputs of the M master neurons in layer L. Thus, the patterns in S_1 would be correctly classified.

Now consider the prototype $\mathbf{\Pi}^q$ of the set of patterns S_2 that are correctly classified by the network at layer $L-1$ (as described earlier). The net input of the master neurons at layer L in response to the prototype $\mathbf{\Pi}^q$ is:

$$n^q_{L_j} = W_{L_j,0} + \sum_{k=1}^{|L-1|} W_{L_j,L-1_k}\pi^q_k$$

$$
\begin{aligned}
&= 2C_j^p + |L-1|\pi_j^q + \sum_{k=1,k\neq j}^{|L-1|} W_{L_j,L-1_k}\pi_k^q \\
&= 2C_j^p + |L-1|\pi_j^q + \sum_{k=1,k\neq j}^{|L-1|} C_j^p\pi_k^p\pi_k^q \\
&= 2C_j^p + |L-1|\pi_j^q + C_j^p\sum_{k=1}^{|L-1|} \pi_k^p\pi_k^q - C_j^p\pi_j^p\pi_j^q \qquad (6)
\end{aligned}
$$

Since $\mathbf{\Pi}^q \neq \mathbf{\Pi}^p$, $-|L-1| \leq \sum_{k=1}^{|L-1|} C_j^p\pi_k^p\pi_k^q \leq |L-1|-2$. Consider a neuron $L-1_\alpha$ $(1 \leq \alpha \leq M)$ such that $\pi_\alpha^q = 1$. From equation 6 :

$$
\begin{aligned}
n_{L_\alpha}^q &= 2C_\alpha^p + |L-1|\pi_\alpha^q + C_\alpha^p\sum_{k=1}^{|L-1|} \pi_k^p\pi_k^q - C_\alpha^p\pi_\alpha^p\pi_\alpha^q \\
&\geq 2C_\alpha^p + |L-1|(1) - C_\alpha^p|L-1| - C_\alpha^p\pi_\alpha^p(1)
\end{aligned}
$$

If $C_\alpha^p = 1$ then $\pi_\alpha^p = -1$ (since the prototype $\mathbf{\Pi}^p$ was not correctly classified). On the other hand, if $C_\alpha^p = -1$ then π_α^p could be either 1 or -1. In either case, after substituting these values in the above equation we see that $n_{L_\alpha}^q \geq 3$. Further for any other neuron $L-1_j$ where $j = 1\ldots M, j \neq \alpha$, $\pi_j^q = -1$:

$$
\begin{aligned}
n_{L_j}^q &= 2C_j^p + |L-1|\pi_j^q + C_j^p\sum_{k=1}^{|L-1|} \pi_k^p\pi_k^q - C_j^p\pi_j^p\pi_j^q \\
&\leq 2C_j^p + |L-1|(-1) + C_j^p(|L-1|-2) - C_j^p\pi_j^p(-1)
\end{aligned}
$$

If $C_j^p = 1$ then $\pi_j^p = -1$ (since the prototype $\mathbf{\Pi}^p$ was not correctly classified). On the other hand, if $C_j^p = -1$ then π_j^p could be either 1 or -1. In either case, after substituting these values in the above equation we see that $n_{L_j}^q \leq -1$. The neuron L_α has the highest net input among all the master neurons L_j $(j = 1\ldots M)$. Thus, $O_{L_\alpha}^q = 1$ and $O_{L_j}^q = -1$ $\forall j = 1\ldots M, j \neq \alpha$, which means that $\mathbf{C}^q = \mathbf{O}^q$.

We have shown that there exists weights for the master neurons of a newly added layer which will reduce the number of misclassifications by at least one. We rely on the algorithm $\mathcal{A}$ to find such weights. Since the training set is finite the algorithm would eventually converge to zero classification errors after adding a sufficient number of layers. □

4.2 Example

The following example illustrates the concepts described in the above proof. Consider the simple dataset S shown in Fig. 4[8]. The first step in the network construction involves training a layer of $M = 3$ master neurons. Let us designate this layer by L^1. One possible set of weights for the master neurons is depicted in Fig. 9. The response of each of master neurons to the patterns in the dataset is summarized in Table 4.

[8] Note that the *MTiling-real* algorithm does not require patterns to be preprocessed.

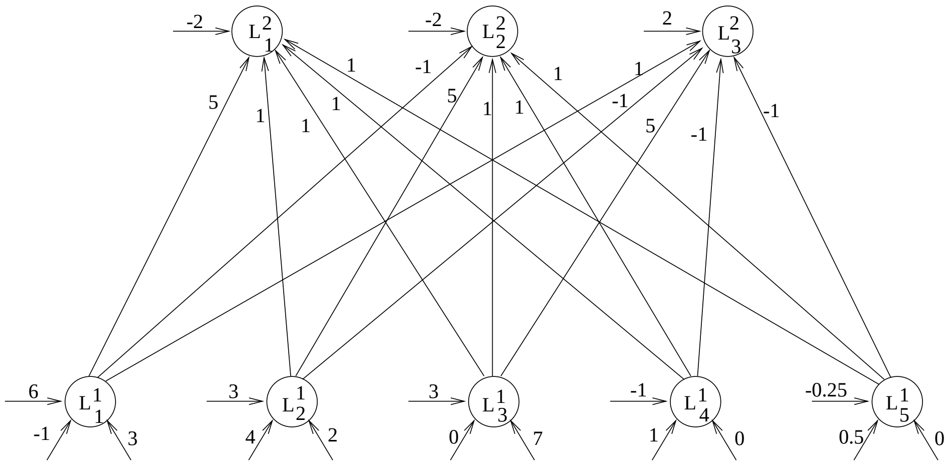

Figure 9: *MTiling-real* Network for the Example Dataset

Table 4: Response of the Layer L^1 Master Neurons

Number	**Net Input**			**Output**		
	$n_{L^1_1}$	$n_{L^1_2}$	$n_{L^1_3}$	$o_{L^1_1}$	$o_{L^1_2}$	$o_{L^1_3}$
1	6	3	3	1	-1	-1
2	9	5	10	-1	-1	1
3	5	7	3	-1	1	-1
4	8	9	10	-1	-1	1
5	7	6	6.5	1	-1	-1

The output of the master neurons is unfaithful with respect to the following two sets of patterns: $S_1 = \{\mathbf{X}^1, \mathbf{X}^5\}$ and $S_2 = \{\mathbf{X}^2, \mathbf{X}^4\}$. Consider the set S_2. $\mathbf{X}^4$ has a dominating attribute $X^4_1 = 1$ ($X^4_1 > X^2_1 = 0$) thus an ancillary neuron with weights $< -(X^4_1)^2 \; X^4_1 \; 0 >$ (i.e., $< -1, 1, 0 >$) makes the layer L^1 faithful with respect to S_2 as per theorem 2. Similarly, a second ancillary neuron with weights set to $< -0.25, 0.5, 0 >$ (see Fig. 9) makes L^1 faithful with respect to S_1. Note that the ancillary neurons are added later (to make the layer faithful) and hence they function as independent TLUs and not as part of the winner-take-all group formed by the master neurons. The output of the patterns in S at layer L^1 is given in Table 5. It can be verified that this output is faithful with respect to all the patterns in S. Note that patterns $\mathbf{X}^1$ and $\mathbf{X}^2$ of the example dataset S are misclassified at layer L^1.

Consider a set $S_1 = \{< 0, 0 >\}$ with corresponding prototype $\mathbf{\Pi}^p =< 1, -1, -1, -1, -1 >$. S_1 is misclassified at layer L^1. The algorithm adds a new layer L^2 of master neurons. The weights of these neurons as per equation 4 are shown in Fig. 9. The net input of the L^2 neurons in response to $\mathbf{\Pi}^p$ is $< -1, -11, 1 >$ and thus the output is $< -1, -1, 1 >$ which shows that the patterns in S_1 are now correctly classified. Next consider a set $S_2 = \{< 1, 0 >\}$ with corresponding prototype $\mathbf{\Pi}^q =< -1, 1, -1, 1, 1 >$. S_2 is correctly classified at layer L^1. The net input of the L^2 neurons in response to $\mathbf{\Pi}^q$ is $< -5, 5, -7 >$ and thus the output is $< -1, 1, -1 >$ which shows that the classification of patterns in S_2 remains unchanged. Similarly, it can be verified that the classification of the patterns $< 1, 1 >$ and $< 0.5, 0.5 >$ also remains unchanged.

Table 5: Response of the Layer L^1 Master and Ancillary Neurons

Number	Net Input Master			Net Input Ancillary		Output				
	$n_{L^1_1}$	$n_{L^1_2}$	$n_{L^1_3}$	$n_{L^1_4}$	$n_{L^1_5}$	$o_{L^1_1}$	$o_{L^1_2}$	$o_{L^1_3}$	$o_{L^1_4}$	$o_{L^1_5}$
1	6	3	3	-1	-0.25	1	-1	-1	-1	-1
2	9	5	10	-1	-0.25	-1	-1	1	-1	-1
3	5	7	3	0	0.25	-1	1	-1	1	1
4	8	9	10	0	0.25	-1	-1	1	1	1
5	7	6	6.5	-0.5	0	1	-1	-1	-1	1

Thus, we have shown that the addition of layer L^2 reduces the number of misclassifications by at least one.

4.3 Pruning Redundant Ancillary Neurons

Each layer of a network constructed using the *MTiling-real* learning algorithm comprises of M master neurons and K (where $K \geq 0$) ancillary neurons. The latter are trained to make the layer faithful with respect to the set of training patterns. During training, if the current layer is *unfaithful* then groups of one or more ancillary neurons are trained for each *unfaithful class* of patterns (i.e., patterns that have exactly the same output at the current layer but belong to different output classes). Ideally, one would expect that each layer contains a minimal number of ancillary neurons necessary to achieve faithfulness. However, in practice, several hidden layers have some redundant ancillary neurons. This can be attributed to the following two reasons: Firstly, owing to the inherent biases of the TLU weight training algorithm $\mathcal{A}$ (used the train the ancillary neurons) and the fact that $\mathcal{A}$ is allowed only a limited training time (typically 500 - 1000 iterations over the training set) more than one group of ancillary neurons might be trained before a faithful representation is attained for a particular unfaithful class. Secondly, as a result of the *locality of training*, whereby each group of ancillary neurons is trained using only a subset of the training patterns, it is possible that not all ancillary neurons are absolutely necessary for faithfulness.

We incorporate a local pruning step in the *MTiling-real* algorithm to remove redundant ancillary neurons. This step is invoked immediately after a layer is made faithful. The check for redundant neurons is simple. Each of the ancillary neurons are systematically dropped (one at a time) and the outputs of the remaining neurons are checked for faithfulness. If the current representation (with an ancillary neuron dropped) is faithful then that ancillary neuron is redundant and is pruned. However, if the current representation is not faithful then the ancillary neuron that is dropped is necessary for faithfulness and hence is brought back. The search for redundant ancillary neurons incurs an additional cost. Let K be the number of ancillary neurons when the layer is first made faithful and $|S|$ be the number of training patterns. The ancillary neurons are dropped one at a time and the outputs of the remaining neurons (including the M master neurons) are checked for faithfulness. The faithfulness test takes $O((M + K) \cdot |S|)$ time and is repeated K times (once for each ancillary neuron). Thus, the total time complexity of the pruning step is $O(K \cdot (M + K) \cdot |S|) \approx O(K^2 \cdot |S|)$. The outputs of the neurons in response to each training pattern are compared for equality during the faithfulness test. These outputs are computed by the *MTiling-real* algorithm when the layer is determined to be faithful and hence do not have to be recomputed. Further, since each neuron only outputs 1 or -1 and the faithfulness test only requires an equality check, the search for redundant neurons can be performed very efficiently. We conducted an experimental study of pruning in *MTiling-real* networks (see [38] for details) and found that the total time spent in searching for and pruning redundant neurons is less than 10% of the total training time. Further, pruning resulted in a moderate to substantial (at times as high as 50%) reduction in network size. Below we illustrate pruning with a simple example.

Consider a training set S comprising of 5 patterns belonging to 3 output classes. Assume that a layer L consisting of $M = 3$ master neurons (say M_1, M_2, and M_3) and $K = 4$ ancillary neurons (say K_1, K_2, K_3, and K_4) is trained to achieve a faithful representation of S. The outputs of the individual neurons in response to the training patterns are depicted in Table 6.

Table 6: Example of Pruning

Output Class	**Master Neurons**			**Ancillary Neurons**			
C^p	$O^p_{M_1}$	$O^p_{M_2}$	$O^p_{M_3}$	$O^p_{K_1}$	$O^p_{K_2}$	$O^p_{K_3}$	$O^p_{K_4}$
I	-1	-1	1	-1	-1	-1	1
II	-1	-1	1	-1	1	-1	1
I	-1	-1	1	-1	-1	1	-1
II	1	-1	-1	-1	1	-1	1
III	1	-1	-1	-1	1	-1	-1

During the pruning step when the ancillary neuron K_1 is dropped, $< M_1,M_2,M_3,K_2,K_3,K_4 >$ is faithful with respect to S. Thus, K_1 is redundant and is pruned. Next, when K_2 is dropped $< M_1, M_2, M_3, K_3, K_4 >$ is not faithful. Thus, K_2 is not redundant and is restored. Next, when K_3 is dropped $< M_1, M_2, M_3, K_2, K_4 >$ (note that K_2 has been restored) is faithful and hence K_3 is pruned. Similarly, we can see that K_4 is not redundant and the final representation of L is $< M_1, M_2, M_3, K_2, K_4 >$.

5 Constructive Learning Algorithms in Practice

The preceding discussion focused on the theoretical proofs of convergence of the *MPyramid-real* and *MTiling-real* constructive learning algorithms. In this section we present results of a few focused experiments that are specifically designed to address the following issues:

1. The convergence proofs presented above 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 $\mathcal{A}$'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 a NP-hard problem [47]. Further, in practice the heuristic algorithms such as the *thermal perceptron algorithm* that are used in constructive learning are only allowed limited training time (say 500 or 1000 epochs). This makes it important to study the convergence of the proposed constructive algorithms in practice.

2. The convergence proofs only guarantee the existence of a set of weights for each newly added neuron (or group of neurons) that will reduce the number of misclassifications by at least one. A network that recruits one neuron to simply memorize each training pattern can trivially attain zero classification errors. A comparison of the actual size of a trained constructive network with the number of patterns in the training set, would at least partially, address this issue of memorization.

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.

5.1 Datasets

A cross-section of datasets having real-valued pattern attributes and patterns belonging to multiple output classes was selected for the experiments. Table 7 summarizes the characteristics of the datasets. **Size** denotes the number of patterns in the dataset, **Inputs** indicates the total number of input attributes (of the unmodified dataset), **Outputs** represents the number of output classes, and **Attributes** describes the type of input attributes of the patterns. The real-world datasets **ionosphere**, **pima**, **segmentation**, and **vehicle** are available at the UCI Machine Learning Repository [34] while the **3-circles** dataset was artificially generated. The **3-circles** dataset comprises of 1800 randomly drawn points from the two dimensional Euclidean space. These points are labeled as belonging to classes 1, 2, and 3 if their distance from the origin is less than 1, between 1 and 2, and between 2 and 3 respectively. Each of these datasets is non-linearly separable.

Table 7: Datasets

Dataset	**Size**	**Inputs**	**Outputs**	**Attributes**
3 concentric circles (**3-circles**)	1800	2	3	real
ionosphere structure (**ionosphere**)	351	34	2	real, int
pima indians diabetes (**pima**)	768	8	2	real,int
image segmentation (**segmentation**)	2310	19	7	real, int
vehicle silhouette (**vehicle**)	846	18	4	int

5.2 Experimental Results

We used the 10-fold cross validation method in our experiments. Each dataset was divided into 10 equal sized folds and 10 independent runs of each algorithm were conducted for each dataset. For the i^{th} run, the i^{th} fold was designated as the test set and the patterns in the remaining 9 folds were used for training. At the end of training the network's generalization was measured on the test set. Individual TLUs were trained using the *thermal perceptron algorithm*. The weights of each neuron were initialized to a random value in the interval $[-1..1]$, the learning rate η was held constant at 1.0, and each neuron was trained for 500 epochs where each epoch involves presenting a set of $|S|$ randomly drawn patterns from the training set S. 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 [6].

Table 8 summarizes the results of experiments designed to test the convergence properties of the constructive learning algorithms. It lists the mean and standard deviation of the network size (the number of hidden and output neurons), the training accuracy, and the test accuracy of the *MPyramid-real* and *MTiling-real* algorithms on the **3-circles** and **ionosphere** datasets. For comparison we include the results of training a single layer network (labeled by *perceptron*) using the *thermal perceptron algorithm*. The training accuracy of the *perceptron* algorithm on both datasets is less that 100% (which confirms the non-linear separability of the datasets). These results show that not only do the constructive algorithms converge to zero classification errors on the training set but they also generalize fairly well on the unseen test data. Further, a comparison of the network sizes with the total number of patterns in each dataset (see table 7) conclusively shows that the constructive learning algorithms are not simply memorizing the training patterns by recruiting one neuron per pattern.

Table 8: Convergence Results

Dataset	**Performance Parameter**	*perceptron*	*MPyramid-real*	*MTiling-real*
3-circles	Network Size	3.0 ± 0.0	6.0 ± 0.0	46.7 ± 3.7
	Train Accuracy %	44.5 ± 5.5	100.0 ± 0.0	100.0 ± 0.0
	Test Accuracy %	41.4 ± 5.4	99.9 ± 0.2	97.0 ± 1.2
ionosphere	Network Size	1.0 ± 0.0	5.0 ± 1.3	5.5 ± 2.3
	Train Accuracy %	97.5 ± 1.0	100.0 ± 0.0	100.0 ± 0.0
	Test Accuracy %	85.4 ± 6.4	90.6 ± 4.3	86.0 ± 6.2

We noticed in our experiments with the other three datasets that the convergence of the algorithms (particularly the *MPyramid-real*) was quite slow. The slow-down was quite pronounced towards the end of the learning cycle where several new layers were added with minimal increase in classification accuracy. Further, at this stage we observed that the generalization accuracy (measured on an independent test set) of the network was deteriorating with the addition of each new layer. This suggests that in an attempt to correctly classify all patterns the algorithms were over-fitting the training data. In practice we are mostly interested in the network's generalization capability. Most backpropagation type learning algorithms use a separate *hold-out* set (distinct from the *test set*) to stop training when over-fitting sets in. In our experiments to measure the generalization performance of the constructive algorithms we used a similar hold-out sample as follows: The 10-fold cross validation was still used but this time during the i^{th} run, the i^{th} fold was designated as the test set, the $i + 1^{th}$ fold as the independent hold-out set, and the remaining eight folds formed the training set. During the network construction process, the accuracy of the network on the hold-out sample was recorded after each new layer was added and trained. At the end of the training (i.e., when the network converged to 100% classification accuracy or the when the network size reached the maximum number of layers — 25 in our case) we pruned the network upto the layer that had the highest accuracy on the hold-out sample. For example, if the trained network network had five layers and the accuracy on the hold-out was recorded as 78%, 82%, 86%, 83%, and 81% at each of the five layers respectively then the layers above layer three were pruned from the network. Note that as a result of the pruning the network's accuracy on the training set will no longer be 100%. At this point we measure the accuracy of the network on the test dataset. It is important to keep in mind that since the test data set is independent of the hold-out set and is not used at all during training the results are not biased or overly optimistic.

Table 9 lists the mean and standard deviation of the network size, training accuracy, and test accuracy of the *MPyramid-real* and *MTiling-real* algorithms for the **pima**, **segmentation**, and **vehicle** datasets. For comparison we include the results of training a single layer network (labeled by *perceptron*) using the *thermal perceptron algorithm*. We see that the pruned networks generated by the *MTiling-real* algorithm are smaller than those generated by the *MPyramid-real* algorithm. This is due to the different network construction schemes adopted by the two algorithms. The *MPyramid-real* algorithm uses the entire training set for training each new layer. Thus, on harder training sets it tends to add several layers of neurons without substantial benefits. On the other hand, the *MTiling-real* algorithm breaks up the dataset into smaller subsets (the *unfaithful classes*). Training of the ancillary neurons on these smaller datasets is considerably simpler. Further, given a faithful representation of the patterns at each layer, the master neurons of the succeeding layer are able to significantly reduce the number of misclassifications. The *MTiling-real* algorithm's focus on smaller subsets for training ancillary neurons might actually prove to be disadvantageous on certain datasets (see for example the **3-circles** in table 8) because it might expend considerable effort in making the current layer faithful.

Table 9: Generalization Results

Dataset	**Performance Parameter**	*perceptron*	*MPyramid-real*	*MTiling-real*
pima	Network Size	1.0 ± 0.0	6.5 ± 4.8	5.7 ± 14.9
	Train Accuracy %	79.3 ± 0.9	79.4 ± 1.9	81.0 ± 4.3
	Test Accuracy %	77.5 ± 3.4	76.8 ± 3.5	77.1 ± 3.5
segmentation	Network Size	7.0 ± 0.0	119.8 ± 34.2	47.1 ± 23.2
	Train Accuracy %	96.0 ± 0.2	94.2 ± 0.9	99.1 ± 1.5
	Test Accuracy %	94.8 ± 2.0	93.0 ± 2.7	99.1 ± 1.7
vehicle	Network Size	4.0 ± 0.0	35.2 ± 28.7	19.4 ± 23.5
	Train Accuracy %	85.5 ± 0.7	87.8 ± 3.3	87.5 ± 4.4
	Test Accuracy %	79.7 ± 5.4	78.2 ± 4.9	77.5 ± 6.2

The striking feature of table 9 is that the test accuracy of the *MPyramid-real* and *MTiling-real* algorithms is almost the same as or even slightly worse than that of the single layer network (except in the case of the **segmentation** dataset where *MTiling-real* performs better). This can be attributed to the following factors: Some of the datasets we used were created at a time when algorithms for training multi-layer networks were just gaining popularity. It is likely that these datasets contain a set of carefully engineered features that were selected by the experts to work well with the algorithms existing at that time [33]. This might result in a scenario where it is not possible to improve the generalization on a particular dataset beyond what is achieved by a single layer network. Additionally, it is possible that these datasets contain irrelevant or noisy attributes that complicate the learning task. Experimental results have shown that using a genetic algorithm based feature feature selection scheme significantly boosts the generalization performance of the *DistAl* learning algorithm [52].

6 Conclusions

Constructive algorithms offer an attractive approach to the automated design of neural networks for pattern classification. In particular, they eliminate the need for the *ad hoc*, and often inappropriate, *a priori* choice of network architecture, they potentially provide a means of constructing networks whose size (complexity) is commensurate with the complexity of the pattern classification task on hand, and they offer natural ways to incorporate prior knowledge to guide learning and to use constructive learning algorithms in the *lifelong* learning framework. We have focused on a family of algorithms that incrementally construct feed forward networks of threshold neurons[9]. 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. We have presented two constructive learning algorithms *MPyramid-real* and *MTiling-real* that extend the *pyramid* and the *tiling* algorithms respectively to handle multi-category classification of patterns that have real-valued attributes. For each of these algorithms we have provided rigorous proofs of convergence to zero classification errors on finite, non-contradictory training sets. This proof technique is sufficiently general (see [37] for an application of this technique to several other constructive learning algorithms).

The convergence of the two algorithms to zero classification errors was established by showing that each modification of the network topology guarantees the existence of weights 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 weights. We do not have a rigorous proof that any of the graceful variants of perceptron learning algorithms can in practice, satisfy the requirements imposed on $\mathcal{A}$, let alone find an *optimal* (in a suitable sense of the term - e.g., so as to yield minimal networks) set of weights. The design of 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. These characteristics of the TLU training algorithm often result in the generation of redundant units during network construction. We have proposed a local pruning strategy that can be used to eliminate redundant neurons (in the *MTiling-real* networks). Experiments with non-linearly separable datasets demonstrate the practical usefulness of the proposed algorithms. On simpler datasets both the *MPyramid-real* and *MTiling-real* algorithms do converge to fairly compact networks with zero classification errors and good generalizability. However, on more difficult tasks convergence is slow. Further, the network might end up memorizing the hard to classify examples thereby resulting in poor generalization. To address this issue we have used an independent *hold-out* set during training to determine the appropriate final network topology. This technique enhances the capability of constructive learning algorithms to generate compact networks with improved generalization. Although it is hard to determine *a priori* which of the two constructive learning algorithms would be suitable for a particular problem, we recommend using the *MTiling-real* al-

[9] Constructive algorithms have also been proposed for the incremental construction of recurrent neural networks (RNN) that learn *finite state automata* from labeled examples. The interested reader is referred to [22, 26] for a discussion on constructive learning of RNN.

gorithm first (during the preliminary analysis) as it tends to have better convergence properties than the *MPyramid-real* algorithm in practice.

Our work has identified the following interesting avenues that merit further investigation :

- *Experimental Comparison of the Performance of Constructive Learning Algorithms*
  The results of our experiments establish the feasibility of using constructive learning algorithms for practical pattern classification. However, in order to assess the true merit of these algorithms it is necessary to conduct a systematic comparison of the performance of these algorithms with others such as *backpropagation*, *cascade correlation*, and *DistAl* on a wide variety of datasets.

- *Analyzing the Bias of Constructive Learning Algorithms*
  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 understanding the advantages and disadvantages of each constructive learning algorithm and consequently in providing some useful recommendations on which algorithm would be more suitable for which types of datasets.

- *Hybrid Learning Algorithms*
  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.

- *Combining Constructive Learning with Feature Selection*
  The generalization performance of learning algorithms can be substantially improved by augmenting them with suitable techniques for selecting a relevant subset of the much larger set of input attributes many of which might be irrelevant or noisy. Several feature subset selection algorithms have been proposed in the pattern recognition literature [42]. The effectiveness of genetic algorithms for feature subset selection in conjunction with the *DistAl* algorithm has been demonstrated in [52].

- *Using Boosting and Error-Correcting Output Codes for Improved Generalization*
  Recent advances in machine learning have resulted in the development of techniques such as *boosting* [17] and *error-correcting output codes* [2] for improving the generalization capability of learning algorithms. Boosting involves training an ensemble of models and using a voting scheme to predict the classification of a formerly unseen pattern. As the name suggests, error-correcting output codes changes the output representation (from the typical $1-of-M$ encoding) to a larger one that facilitates error correction. An application of these techniques in the constructive learning framework is clearly of interest.

- *Knowledge Extraction from Trained Constructive Neural Networks*
  The incorporation of domain specific knowledge into an initial network topology and its subsequent refinement using constructive learning has been studied in [14, 35, 36]. The question now is whether we can use some of the existing strategies (see for example [9]) or design suitable new methods for extracting the learned knowledge from a trained constructive network.

References

[1] E. Alpaydin. Gal: Networks that grow when they learn and shrink when they forget. Technical Report TR91-032, International Computer Science Institute, Berkeley, 1991.

[2] G. Bakiri and T. Dietterich. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[3] E. Baum. A proposal for more powerful learning algorithms. *Neural Computation*, 1(2):201–207, 1989.

[4] K. Bennett and O. Mangasarian. Neural network training via linear programming. Technical Report Technical Report 948, Department of Computer Science, University of Wisconsin-Madison, 1990.

[5] N. Bose and A. Garga. Neural network design using voronoi diagrams. *IEEE Transactions on Neural Networks*, 4(5):778–787, 1993.

[6] N. Burgess. A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5(1):59–66, 1994.

[7] C-H. Chen, R. Parekh, J. Yang, K. Balakrishnan, and V. Honavar. Analysis of decision boundaries generated by constructive neural network learning algorithms. In *Proceedings of WCNN'95, July 17-21, Washington D.C.*, volume 1, pages 628–635, 1995.

[8] T. Cormen, C. Leiserson, and Rivest R. *Introduction to Algorithms*. MIT Press, Cambridge, 1991.

[9] M. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Department of Computer Science, University of Wisconsin, Madison, WI, 1996.

[10] J. Dayhoff. *Neural Network Architectures: An Introduction*. Van Nostrand Reinhold, New York, 1990.

[11] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202, San Fransisco, CA, 1995.

[12] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[13] S. Fahlman and C. Lebiere. The cascade correlation learning algorithm. In D. Touretzky, editor, *Neural Information Systems 2*, pages 524–532. Morgan-Kauffman, 1990.

[14] J. Fletcher and Z. Obradović. Combining prior symbolic knowledge and constructive neural netwo rk learning. *Connection Science*, 5(3,4):365–375, 1993.

[15] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209, 1990.

[16] M. Frean. A thermal perceptron learning rule. *Neural Computation*, 4:946–957, 1992.

[17] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning algorithms and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[18] J. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, 1981.

[19] S. Gallant. Perceptron based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, June 1990.

[20] S. Gallant. *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, MA, 1993.

[21] S. Geva and J. Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Transactions on Neural Networks*, 3(4):621–624, 1992.

[22] C. Giles, D. Chen, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chung Lee, and M Goudreau. Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 6(4):829–836, 1997.

[23] V. Honavar. *Generative Learning Structures and Processes for Generalized Connectionist Networks.* PhD thesis, University of Wisconsin, Madison, 1990.

[24] V. Honavar and L Uhr. Generative learning structures for generalized connectionist networks. *Information Sciences*, 70(1-2):75–108, 1993.

[25] T. Hrycej. *Modular Learning in Neural Networks.* Wiley, New York, 1992.

[26] S. Kremer. Comments on "constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 7(4):1047–1049, 1996.

[27] Tin-Yau Kwok and Dit-Yan Yeung. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8(5):1131–1148, 1997.

[28] Tin-Yau Kwok and Dit-Yan Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 1999. (To appear).

[29] M. Marchand, M. Golea, and P. Rujan. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6):487–492, 1990.

[30] F. Mascioli and G. Martinelli. A constructive algorithm for binary neural networks: The oil-spot algorithm. *IEEE Transactionson Neural Networks*, 6(3):794–797, 1995.

[31] K. Mehrotra, C. Mohan, and S. Ranka. *Elements of Artificial Neural Networks.* MIT Press, Cambridge, Massachusetts, 1997.

[32] M. Mézard and J. Nadal. Learning feed-forward networks: The tiling algorithm. *J. Phys. A: Math. Gen.*, 22:2191–2203, 1989.

[33] R. Mooney, J. Shavlik, G. Towell, and Alan Gove. An experimental comparison of symbolic and connectionist learning algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 775–780. Morgan Kauffman, 1989.

[34] P. Murphy and D. Aha. Repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA, 1994.

[35] D. W. Opitz and J. W. Shavlik. Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge-Based Systems*, 8(6):301–311, 1995.

[36] R. Parekh and V. Honavar. Constructive theory refinement in knowledge based neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'98)*, pages 2318–2323, Anchorage, AK, 1998.

[37] R. Parekh, J. Yang, and V. Honavar. Constructive neural network learning algorithms for multi-category real-valued pattern classification. Technical Report ISU-CS-TR97-06, Department of Computer Science, Iowa State University, 1997.

[38] R. Parekh, J. Yang, and V. Honavar. Pruning strategies for constructive neural network learning algorithms. In *Proceedings of the IEEE/INNS International Conference on Neural Networks, ICNN'97*, pages 1960–1965, 1997.

[39] H. Poulard. Barycentric correction procedure: A fast method of learning threshold units. In *Proceedings of WCNN'95, July 17-21, Washington D.C.*, volume 1, pages 710–713, 1995.

[40] L. Prechelt. Investigating the cascor family of learning algorithms. *Neural Networks*, 10(5):885–896, 1997.

[41] R. Reed. Pruning algorithms — a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[42] B. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, New York, 1996.

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

[44] P. Rujàn and M. Marchand. Learning by minimizing resources in neural networks. *Complex Systems*, 3:229–241, 1989.

[45] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, volume 1 (Foundations). MIT Press, Cambridge, Massachusetts, 1986.

[46] J. Saffery and C. Thornton. Using stereographic projection as a preprocessing technique for upstart. In *Proceedings of the International Joint Conference on Neural Networks*, pages II 441–446. IEEE Press, July 1991.

[47] K-Y. Siu, V. Roychowdhury, and T. Kailath. *Discrete Neural Computation - A Theoretical Foundation.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[48] F. Śmieja. Neural network constructive algorithms: Trading generalization for learning efficiency? *Circuits, Systems, and Signal Processing*, 12(2):331–374, 1993.

[49] S. Thrun. Lifelong learning: A case study. Technical Report CMU-CS-95-208, Carnegie Mellon University, 1995.

[50] G. G. Towell, J. W. Shavlik, and M. O. Noordwier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA, 1990.

[51] J. Yang and V. Honavar. Experiments with the cascade-correlation algorithm. *Microcomputer Applications*, 1998. (To appear).

[52] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems (Sepcial Issue on Feature Transformation and Subset Selection)*, 13(2):44–49, 1998.

[53] J. Yang, R. Parekh, and V. Honavar. MTiling - a constructive neural network learning algorithm for multi-category pattern classification. In *Proceedings of the World Congress on Neural Networks'96*, pages 182–187, San Diego, 1996.

[54] J. Yang, R. Parekh, and V. Honavar. DistAl: An inter-pattern distance-based constructive learning algorithm. *Intelligent Data Analysis*, 1999. (To appear).

[55] D. Yeung. Constructive neural networks as estimators of bayesian discriminant functions. *Pattern Recognition*, 26(1):189–204, 1993.