MUpstart - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification

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

Abstract

Constructive learning algorithms offer an approach for dynamically constructing near-minimal neural network architectures for pattern classification tasks. Several such algorithms proposed in the literature are shown to converge to zero classification errors on finite non-contradictory datasets. However, these algorithms are restricted to two-category pattern classification and (in most cases) they require the input patterns to have binary (or bipolar) valued attributes only. We present a provably correct extension of the Upstart *algorithm to handle multiple output classes and real-valued pattern attributes. Results of experiments with several artificial and real-world datasets demonstrate the feasibility of this approach in practical pattern classification tasks and also suggest several interesting directions for future research.*

1. Introduction

Multi-layer networks of threshold logic units (also called threshold neurons or TLU) or multi-layer perceptrons (MLP) offer an attractive framework for the design of pattern classification and inductive knowledge acquisition systems for a number of reasons including: potential for parallelism and fault tolerance; significant representational and computational efficiency that they offer over disjunctive normal form (DNF) functions and decision trees [6]; 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 computes the binary hard-limiting function of the weighted sum of its inputs. Assuming that the patterns are drawn from an N-dimensional Euclidean space, the output O^p, of a TLU with an N-element weight vector $\mathbf{W}$, in response to a pattern $\mathbf{X}^p$, is 1 if $\mathbf{W} \cdot \mathbf{X}^p \geq 0$ and 0 otherwise. A TLU implements a $(N-1)$-dimensional hyperplane given by $\mathbf{W} \cdot \mathbf{X} = 0$ which partitions the N-dimensional Euclidean pattern space into two regions (or two classes). Given a set of *examples* $S = S^+ \cup S^-$ where S^+ and S^- represent patterns with target outputs 1 and 0 respectively, it is the goal of a *perceptron training* algorithm to attempt to find a weight vector $\hat{\mathbf{W}}$ such that $\forall \mathbf{X}^p \in S^+$, $\hat{\mathbf{W}} \cdot \mathbf{X}^p \geq 0$ and $\forall \mathbf{X}^p \in S^-$, $\hat{\mathbf{W}} \cdot \mathbf{X}^p < 0$. If such a weight vector ($\hat{\mathbf{W}}$) exists for the pattern set S then S is said to be *linearly separable*. The *Perceptron* weight update rule [11]: $\mathbf{W} \leftarrow \mathbf{W} + \eta(C^p - O^p)\mathbf{X}^p$ (where C^p is the desired output for pattern $\mathbf{X}^p$ and $\eta > 0$ is the learning rate) is an iterative algorithm for determining $\hat{\mathbf{W}}$ if one exists. However when S is not linearly separable, the *Perceptron* algorithm behaves 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., the *Pocket algorithm* with *ratchet modification* [5], the *Thermal perceptron algorithm* [4], and the *Barycentric correction procedure* [10]) are designed to find a reasonably good weight vector that correctly classifies a large fraction of the training set S when S is not linearly separable and to converge to zero classification errors when S is linearly separable. For a detailed comparison of the algorithms for training TLUs see [13].

When S is not linearly separable, a multi-layer network of TLUs is needed to learn a non-linear decision boundary that correctly classifies all the training examples. We focus on *constructive* or *generative* learning algorithms that incrementally construct networks of threshold neurons to correctly classify a given (typically non-linearly separable) pattern set. Constructive learning algorithms are character-

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

ized by their potential to construct near minimal network architectures, guaranteed convergence, and ability to provide natural trade offs among performance measures such as training time, generalization ability, and network size. A number of constructive learning algorithms for 2-category pattern classification have been proposed in the literature — *Tower*, *Pyramid* [5], *Tiling* [7], *Upstart* [3], and *Perceptron Cascade* [1]. These 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 [2] for an analysis (in geometrical terms) of the decision boundaries generated by some of these constructive learning algorithms. The convergence proof of each algorithm is based on the ability of the TLU weight training algorithm to find a weight setting for each newly added neuron or neurons such that the number of pattern misclassifications is reduced by at least one each time a neuron (or a set of neurons) is added and trained and the network's outputs are recomputed. Choices for an appropriate TLU weight training algorithm ($\mathcal{A}$) include the *Pocket algorithm* with *ratchet modification*, the *Thermal perceptron algorithm*, and the *Barycentric correction procedure* .

Pattern classification tasks often require assigning patterns to one of M ($M > 2$) classes. Although in principle, an M-category classification task can be reduced to an equivalent set of M 2-category classification tasks (each with its own training set constructed from the given M-category training set), a better approach might be one that takes into account the inter-relationships between the M output classes. In the case of most constructive learning algorithms, extensions to multiple output classes have not been explored. In other cases, only some preliminary ideas (not supported by detailed theoretical or experimental analysis) for possible multi-category extensions of 2-category algorithms are available in the literature.

For pattern sets that involve multiple output classes, training can be performed either *independently* or by means of the *winner take all* (WTA) strategy [6]. In the former, each output neuron is trained independently of the others using one of the TLU weight training algorithms mentioned earlier. The fact that the membership of a pattern in one class precludes its membership in any other class can be exploited to compute the outputs using the WTA strategy wherein, for any pattern, the output neuron with the highest net input is assigned an output of 1 and all other neurons are assigned outputs of 0. In the case of a tie for the highest net input all neurons are assigned an output of 0, thereby rendering the pattern incorrectly classified. It is thus of interest to apply the WTA strategy for computing the outputs in constructive learning algorithms. For details on the adaptation of the TLU training algorithms to the WTA strategy see [13].

Additionally, practical classification tasks often involve patterns with real-valued attributes. The TLU weight training algorithms like the *Pocket algorithm* with *ratchet modification*, the *Thermal perceptron algorithm*, and the *Barycentric correction procedure* do handle patterns with real-valued attributes. However, extensions of the constructive learning algorithms to handle patterns with real-valued attributes have only been studied for the *Upstart* [12] and the *Perceptron Cascade* [1] algorithms.

We present MUpstart, an extension of the *Upstart* algorithm. MUpstart is a provably correct constructive learning algorithm that handles multiple output classes, real-valued attributes, and facilitates both independent and WTA training of the output neurons. Preliminary experiments on several *artificial* and *real-world* datasets demonstrate the practical applicability of this algorithm.

2. The MUpstart Algorithm

The 2-category *Upstart* algorithm [3] 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 outputs. Therefore, in what follows, we take an alternative approach using a single hidden layer instead of a binary tree. Further, to handle patterns with real-valued attributes we project the input patterns on to a parabolic surface by appending an additional attribute to each pattern. This attribute takes on a value equal to the sum of squares of the values of all other attributes in the pattern. This idea of considering projections of input patterns was first described in [12].

The following notation is used in the description of the MUpstart algorithm: N is the number of pattern attributes; M is the number of output neurons (for 2-category tasks $M = 1$); I is the input layer index; $1, 2, \ldots L$ are the indices for the other layers; U_A is the number of neurons in layer A; $A_1, A_2, \ldots, A_{U_A}$ refer to the individual neurons in layer A; W_{A_i,B_j} is the weight between neuron A_i and B_j; $W_{A_i,0}$ is the threshold for neuron A_i; $\mathbf{X}^p = < X_0^p, X_1^p, \ldots, X_N^p >$, $X_0^p = 1$ for all p, is the augmented pattern vector; $\hat{\mathbf{X}}^p = < X_0^p, X_1^p, \ldots, X_N^p, X_{N+1}^p >$ (where $X_{N+1}^p = \sum_{i=1}^{N}(X_i^p)^2$) is the projected pattern vector; $n_{A_j}^p$ and $O_{A_j}^p$ are the net input and observed output respectively for neuron A_j in response to pattern $\mathbf{X}^p$; $\mathbf{C}^p$ is the target output vector; and e_A is the number of misclassifications at layer A.

A pattern is said to be correctly classified at layer A when

$\mathbf{C}^p = \mathbf{O}^p_{\mathcal{A}}$. As is standard in neural networks literature we will assume that the input layer (I) neurons are *linear* neurons with a single input (whose weight is set to 1). In the case of the MUpstart algorithm, the hidden and output layer neurons are TLUs implementing the binary threshold function.

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

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

The daughter neuron is trained using the algorithm $\mathcal{A}$. It is then connected to each neuron in the output layer and the output weights are retrained. The resulting network is shown in Fig. 1.

2.1. Algorithm

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

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

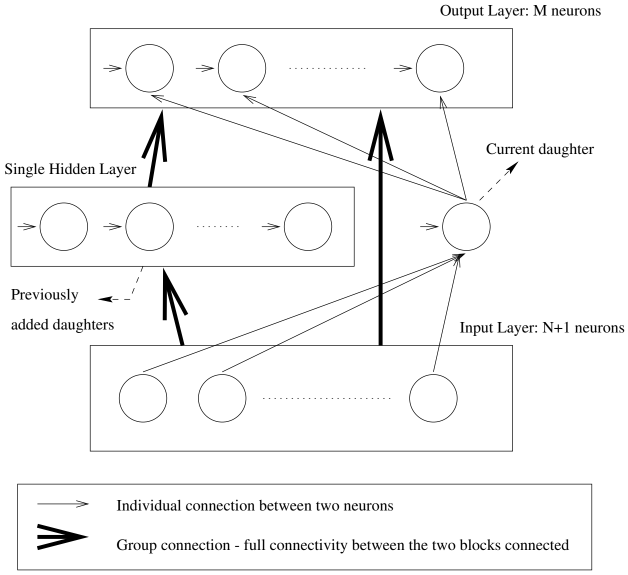

Figure 1. MUpstart **Network**

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

2.2. Convergence Proof

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

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

Consider the following weight setting for the daughter neuron:

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

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

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

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

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

Consider the following weight setting for connections between each output layer neuron and the newly trained X daughter:

$$W_{L_j,X} = 2(C_j^p - O_{L_j}^p)\lambda \text{ for } j = 1 \ldots M$$

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

$$\begin{aligned} n_{L_j}^p &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^p + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^p \\ &\quad +2(C_j^p - O_{L_j}^p)\lambda O_X^p \\ &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^p + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^p \\ &\quad +2(C_j^p - O_{L_j}^p)\lambda(1) \end{aligned} \quad (2)$$

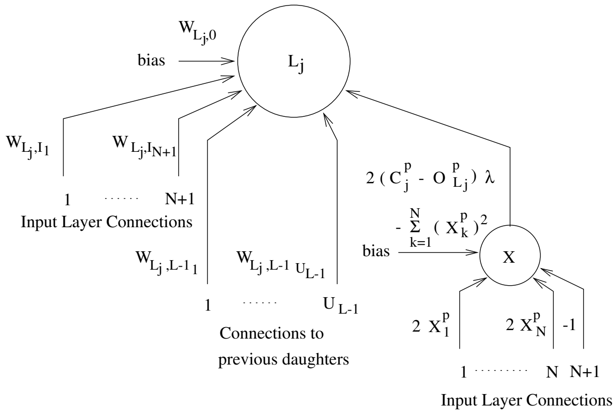

Figure 2. Weight setting for neuron L_j

By the definition of λ we know that

$$-\lambda \leq \max_j [W_{L_j,0} + \sum_{k=1}^{N+1} W_{L_j,I_k} X_k^p + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^p] \leq \lambda \quad (3)$$

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

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

$$\begin{aligned} n_{L_j}^q &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^q + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^q \\ &\quad +2(C_j^p - O_{L_j}^p)\lambda O_X^q \\ &= W_{L_j,0} + \sum_{i=1}^{N+1} W_{L_j,I_i} X_i^q \\ &\quad + \sum_{k=1}^{U_{L-1}} W_{L_j,L-1_k} O_{L-1_k}^q \end{aligned} \quad (4)$$

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

A similar proof can be presented for the case when a wrongly-off corrector (i.e., a Y daughter) is added to the hidden layer. Thus, we see that the addition of a daughter ensures that the number of misclassified patterns is reduced by at least one. Since the number of patterns in the training set is finite, the number of errors is guaranteed to eventually become zero. Further it is easy to show that the same weight setting for the daughter and output neurons can be used to show convergence even when the outputs are computed according to the WTA strategy [9].

We have thus proved the convergence of the MUpstart algorithm. □

3. Experimental Results

We have conducted experiments on the MUpstart algorithm using a variety of *artificial* and *real-world* datasets. Our experiments were aimed at assessing three important issues viz. convergence to zero classification errors, generalization performance, and network size. Table 1 summarizes the properties of the datasets where **Tr**, **Te**, **N**, **M**, and **A** denote the size of the training set, the size of the test set, the number of inputs, the number of output neurons, and the attribute type (*r* - real, *i* - integer, *b* - binary/bipolar), respectively. The 3 category 5 bit random function (**r5**) randomly assigns the 32 training patterns to one of three output classes. Training was performed for 5 different random functions. The 3 concentric circles dataset (**3c**) considers points belonging to three concentric circles centered at the origin and having radii 2, 4, and 6 respectively. Points were generated at random and were assigned to one of three output classes depending on the distance ($d(x, O)$) of the point (x) from the origin (O): $\forall x,\ 0 \leq d(x, O) \leq 2 \Longrightarrow x \in$ class 1 ; $2 < d(x, O) \leq 4 \Longrightarrow x \in$ class 2 ; and $4 < d(x, O) \leq 6 \Longrightarrow x \in$ class 3 . The real-world datasets viz. ionosphere structure (**ionosphere**), image segmentation (**segmentation**), and wine recognition (**wine**) were taken from the *UCI Machine Learning Repository* [2].

Dataset	**Tr**	**Te**	**N**	**M**	**A**
r5	32	—	5	3	b
3-circles	900	900	2	3	r
ionosphere	234	117	34	1	i,r
segmentation	210	2100	19	7	i,r
wine	120	58	13	3	i,r

Table 1. Datasets

[2] [http://www.ics.uci.edu/AI/ML/MLDBRepository.html]

The individual neurons were trained for 500 epochs using the *Thermal perceptron algorithm* using the WTA output strategy for datasets with $M > 2$ output categories. The average network size and generalization accuracy over 25 runs of the MUpstart algorithm for each dataset are reported in Table 2. Training was stopped if the algorithm did not converge even after adding 100 daughter neurons. Convergence was not obtained on the **wine** dataset for any run. However, the algorithm did converge when the *Barycentric correction procedure* was used instead of the *Thermal perceptron algorithm* for training individual TLUs.

Dataset	**Network Size**	**Test Accuracy**
r5	10.64±0.57	—
3-circles	19.00±15.53	99.03±0.76
ionosphere	3.04±0.45	94.12±1.89
segmentation	14.76±1.94	86.77±1.47
wine	14.76±10.17	90.48±3.76

Table 2. Experiments with MUpstart

These results demonstrate the applicability of the MUpstart algorithm on practical pattern classification tasks. We see that the algorithm converges to zero classification errors on the different training sets. A comparison of the average network size with the number of training patterns clearly demonstrates that the algorithm is not simply memorizing patterns by assigning one daughter neuron per pattern. Further, the networks constructed exhibit good generalization on test sets. For a detailed analysis of the different performance issues and a comparison of the MUpstart algorithm with other constructive learning algorithms see [9].

4. Discussion

Constructive neural network learning algorithms offer a powerful approach to inductive learning for pattern classification applications. This paper has developed MUpstart, a provably convergent extension of the *Upstart* algorithm to handle multi-category classification and real-valued pattern attributes. Experiments have demonstrated the feasibility of this algorithm on practical pattern classification tasks. The convergence proof for MUpstart can be easily adapted to establish the convergence of a multi-category extension of the *Perceptron Cascade* algorithm [1, 8, 9].

The convergence of MUpstart 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 the error before such a modification is made and assuming that weight modification algorithm $\mathcal{A}$ used 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 $\mathcal{A}$, let alone find an *optimal* set of weights (in some suitable well-defined sense of the term - e.g., so as to yield minimal networks). The design of suitable threshold neuron 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. Detailed theoretical and experimental analysis of the performance of single threshold neuron training algorithms is in progress [13].

Since our primary focus was on a provably convergent multi-category extension of the *Upstart* algorithm, we have not addressed a number of important issues in this paper. The design choices that determine the output unit for which a corrector daughter is added, the type of the daughter unit, how the daughter unit is trained, etc. impose a set of inductive and representative biases on the MUpstart algorithm. A systematic characterization of these biases would be useful in guiding the design of improved constructive algorithms. Comparative analysis of the performance of various constructive algorithms on a broad range of datasets is currently in progress. An improvement in generalization performance might be achieved by using *cross-validation* based training where the addition of further daughter neurons is stopped when the classification accuracy on a test set begins to deteriorate. Pre-processing techniques such as *normalizing* the pattern vectors and *feature selection* and post-processing methods such as *network pruning* that might assist in obtaining compact networks with good generalization ability merit further investigation.

References

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

[2] 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.

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

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

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

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

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

[8] R. G. Parekh, J. Yang, and V. G. Honavar. Constructive neural network learning algorithms for multi-category classification. Technical Report ISU-CS-TR95-15a, Department of Computer Science, Iowa State University, 1995.

[9] R. G. Parekh, J. Yang, and V. G. 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.

[10] 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.

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

[12] 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.

[13] J. Yang, R. G. Parekh, and V. G. Honavar. Empirical comparison of graceful variants of the perceptron learning algorithm on non-separable data sets. In preparation, 1997.