# CONSTRUCTIVE LEARNING AND STRUCTURAL LEARNING

Trainable pattern classifiers find a broad range of applications in data mining and knowledge discovery (1,2), intelligent agents (3,4), diagnosis (5), computer vision (6), and automated knowledge acquisition (2,7–9) from data. Multilayer networks of threshold logic units (TLU) (10–15) offer an attractive framework for the design of trainable pattern classification systems for a number of reasons including potential for parallelism and fault and noise tolerance; significant representational and computational efficiency over disjunctive normal form (DNF) expressions and decision trees (11); and simpler digital hardware implementations than their continuous counterparts, such as sigmoid neurons used in networks trained with an error backpropagation algorithm (16,17).

A TLU implements an $(N - 1)$-dimensional hyperplane which partitions an $N$-dimensional Euclidean pattern space into two regions. A single TLU neural network is sufficient to classify patterns in two classes if they are *linearly separable.* A number of learning algorithms that are guaranteed to find a TLU weight setting that correctly classifies a linearly separable pattern set have been proposed in the literature (11,18–24). However, when the given set of patterns is not linearly separable, a multilayer network of TLUs is needed to learn a complex decision boundary that is necessary to correctly classify the training examples.

Broadly speaking, there are two approaches to the design of multilayer neural networks for pattern classification:

1. *A priori Fixed Topology Networks.* The number of layers, the number of hidden neurons in each hidden layer, and the connections between each neuron are defined *a priori* for each classification task. This is done on the basis of problem-specific knowledge (if available), or in *ad hoc* fashion (requiring a process of trial and error). Learning in such networks usually amounts to search-

ing (typically in an error gradient guided manner) for a suitable setting of numerical parameters and weights in a weight space defined by the choice of the network topology.

2. *Adaptive Topology Networks.* The topology of the target network is determined dynamically by introducing new neurons, layers, and connections in a controlled fashion using generative or constructive learning algorithms. In some cases, pruning mechanisms that discard redundant neurons and connections are used in conjunction with the network construction mechanisms (25,26).

Structural Learning Algorithms for Neural Networks offer the following advantages over the conventional backpropagation style learning approaches (12,27,28):

1. *Limitations of Learning by Weight Modification Alone Within an Otherwise a priori Fixed Network Topology.* Weight modification algorithms typically search for a solution weight vector that satisfies some desired performance criterion (e.g., classification error). For this approach to be successful, such a solution must lie within the weight-space being searched, and the search procedure employed must in fact, be able to locate it. This means that unless the user has adequate problem-specific knowledge that could be brought to bear upon the task of choosing an adequate network topology, the process is reduced to one of trial and error. Constructive algorithms can potentially offer a way around this problem by extending the search for a solution, in a controlled fashion, to the space of network topologies.
2. *Complexity of the Network Should Match the Intrinsic Complexity of the Classification Task.* It is desirable that a learning algorithm construct networks whose complexity (as measured in terms of relevant criteria such as number of nodes, number of links, connectivity, etc.) is commensurate with the intrinsic complexity of the classification task (implicitly specified by the training data). Smaller networks yield efficient hardware implementations. Everything else being equal, the more compact the network, the more likely it is that it exhibits better generalization properties. Constructive algorithms can potentially discover near-minimal networks for correct classification of a given dataset.
3. *Estimation of Expected Case Complexity of Pattern Classification Tasks.* Many pattern classification tasks are known to be computationally hard. However, little is known about the *expected* case complexity of classification tasks that are encountered, and successfully solved, by living systems, primarily because it is difficult to mathematically characterize the statistical distribution of such problem instances. Constructive algorithms, if successful, can provide useful empirical estimates of expected case complexity of real-world pattern classification tasks.
4. *Trade-Offs Among Performance Measures.* Different constructive learning algorithms offer natural means of trading off certain subsets of performance measures (e.g., learning time) against others (network size, generalization accuracy).
5. *Incorporation of Prior Knowledge.* Constructive algorithms provide a natural framework for exploiting problem-specific knowledge (e.g., in the form of production rules) into the initial network configuration or heuristic knowledge (e.g., about the general topological constraints on the network) into the network construction algorithm.

Several constructive algorithms that incrementally construct networks of threshold neurons for two-category pattern classification tasks have been proposed in the literature. These include the tower (29,30), pyramid (30), tiling (31), upstart (32), perceptron cascade (33), and sequential (34) algorithms. Recently, provably correct extensions of these algorithms to handle multiple output classes and real-valued pattern attributes were proposed [see (12–14)]. With the exception of the sequential learning algorithm, these constructive learning algorithms are based on the idea of transforming the hard task of determining the necessary network topology and weights to two subtasks: (a) incremental addition of one or more threshold neurons to the network when the existing network topology fails to achieve the desired classification accuracy on the training set. (b) training the added threshold neuron(s) using some variant of the perceptron training algorithm [e.g., the pocket algorithm (11)] to improve the classification accuracy of the network. In the case of the sequential learning algorithm, hidden neurons are added and trained by an appropriate weight-training rule to exclude patterns belonging to the same class from the rest of the pattern set.

The constructive algorithms differ in terms of their choices regarding restrictions on input representation (e.g., binary, bipolar, or real-valued inputs); when to add a neuron; where to add a neuron; connectivity of the added neuron; weight initialization for the added neuron; how to train the added neuron (or a subnetwork affected by the addition); and so on. The interested reader is referred to Ref. 10 for an analysis (in geometrical terms) of the decision boundaries generated by some of these constructive learning algorithms. Each of these algorithms can be shown to converge to networks that yield zero classification errors on any given training set wherein the patterns belong to one of two classes (i.e., two-category classification). To keep the discussion that follows focused, we use a specific constructive algorithm—DistAl—to illustrate the key ideas.

DistAl can be viewed as a variant of the instance-based, nearest-neighbor, and radial-basis function-based approaches to pattern classification. DistAl replaces the iterative weight update of neurons that is typically used in constructive learning algorithms by a comparison of pair-wise distances among the training patterns. Because the interpattern distances are computed only once during the execution of the algorithm, our approach achieves a significant speed advantage over other constructive learning algorithms.

## DistAl

DistAl differs from other constructive learning algorithms mentioned above in two respects: Firstly, it uses *spherical* threshold units (a variant of the TLU) as hidden neurons. The regions that are defined (or separated) by TLUs are unbounded. This motivates us to use spherical threshold units that cover locally bounded regions (8). A spherical threshold neuron $i$ has associated with it a weight vector $\boldsymbol{W}_i$, two thresholds $-\ \theta_{i,\text{low}}$ and $\theta_{i,\text{high}}$, and a suitably defined distance metric

$d$. It computes the distance $d(\boldsymbol{W}_i, \boldsymbol{X}^p)$ between a given input pattern $\boldsymbol{X}^p$ and $\boldsymbol{W}_i$. The corresponding output $o_i^p = 1$ if $\theta_{i,\text{low}} \leq d(\boldsymbol{W}_i, \boldsymbol{X}^p) \leq \theta_{i,\text{high}}$ and 0 otherwise. The spherical neuron thus identifies a cluster of patterns that lie in the region between two concentric hyperspherical regions. $\boldsymbol{W}_i$ represents the common center and $\theta_{i,\text{low}}$ and $\theta_{i,\text{high}}$ respectively represent the boundaries of the two regions.

Secondly, DistAl does not use an iterative algorithm for finding the weights and the thresholds. Instead, it computes the interpattern distance once between each pair of patterns in the training set and determines the weight values for hidden neurons by a greedy strategy (one that attempts to correctly classify as many patterns as possible with the introduction of each new hidden neuron). The weights and thresholds are then set without the computationally expensive iterative process (see the section on Network Construction for details).

The use of a one-time interpattern distance calculation instead of a (usually) iterative, expensive, and time-consuming perceptron training procedure makes the proposed algorithm significantly faster than most other constructive learning algorithms. In fact, the time and space complexities of DistAl can be shown to be polynomial in the size of the training set.

### Distance Metrics

Each hidden neuron introduced by DistAl essentially represents clusters of patterns that fall in the region bounded by two concentric hyperspherical regions in the pattern space. The weight vector of the neuron defines the center of the hyperspherical regions and the thresholds determine the boundaries of the regions (relative to the choice of the distance metric used). The choice of an appropriate distance metric for the hidden layer neurons is critical to achieving a good performance. Different distance metrics represent different notions of distance in the pattern space. They also impose different inductive biases (7,8) on the learning algorithm. Consequently, many researchers have investigated the use of alternative distance functions for instance-based learning (6,35–38). The number and distribution of the clusters that result from specific choices of distance functions is a function of the distribution of the patterns as well as the clustering strategy used. Because it is difficult to identify the best distance metric in the absence of knowledge about the distribution of patterns in the pattern space, we chose to explore a number of different distance metrics proposed in the literature.

The distance between two patterns is often skewed by attributes that have high values. *Normalization* of individual attributes overcomes this problem in the distance computation. Normalization can be achieved by dividing each pattern attribute by the range of possible values for that attribute or by four times the standard deviation for that attribute (38).

Normalization also allows attributes with nominal and/or missing values to be considered in distance computation. The distance for attributes with nominal values (say with attribute values $x$ and $y$) is computed as follows (38):

- Overlap: $d_{ol}(x, y) = 0$ if $x = y$; 1 otherwise.
- Value difference:

$$d_{vd}(x, y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q$$

where

$N_{a,x}(N_{a,y})$ is the number of patterns in the training set that have value $x(y)$ for attribute $a$

$N_{a,x,c}(N_{a,y,c})$ is the number of patterns in the training set that have value $x(y)$ for attribute $a$ and output class $c$

$C$ is the number of output classes

$q$ is a constant (Euclidean, 2; Manhattan, 1)

If there is a missing value in either of the patterns, the distance for that component (of the entire pattern vector) is taken to be 1.

Let $\boldsymbol{X}^p = [X_1^p, \cdots, X_n^p]$ and $\boldsymbol{X}^q = [X_1^q, \cdots, X_n^q]$ be two pattern vectors. Let $\max_i$, $\min_i$ and $\sigma_i$ be the maximum, minimum, and the standard deviation of values of the $i$th attribute of patterns in a dataset, respectively. Then the distance between $\boldsymbol{X}^p$ and $\boldsymbol{X}^q$, for different choices of the distance metric $d$ is defined as follows:

1. Range, value-difference based Euclidean:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{X_i^p - X_i^q}{\max_i - \min_i}\right)^2 \text{ or } d_{vd}(X_i^p, X_i^q)^2\right]}$$

2. Range, value-difference based Manhattan:

$$\frac{1}{n}\sum_{i=1}^{n}\left[\frac{X_i^p - X_i^q}{\max_i - \min_i} \text{ or } d_{vd}(X_i^p, X_i^q)\right]$$

3. Range, value-difference based maximum value:

$$\max_i \left[\frac{|X_i^p - X_i^q|}{\max_i - \min_i} \text{ or } d_{vd}(X_i^p, X_i^q)\right]$$

Similarly, $4 * \sigma_i$ can be used instead of $\max_i - \min_i$ for standard-deviation-based metrics, and $d_{ol}(X_i^p, X_i^q)$ can be used instead of $d_{vd}(X_i^p, X_i^q)$ for overlap-based metrics in above formulas.

4. Dice coefficient:

$$1 - \frac{2\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2}$$

5. Cosine coefficient:

$$1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sqrt{\sum_{i=1}^{n}(X_i^p)^2 \cdot \sum_{i=1}^{n}(X_i^q)^2}}$$

6. Jaccard coefficient:

$$1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2 - \sum_{i=1}^{n} X_i^p X_i^q}$$

7. Camberra:

$$\sum_{i=1}^{n} \frac{|X_i^p - X_i^q|}{|X_i^p + X_i^q|}$$

### Network Construction

Let $S = \{\boldsymbol{X}^1, \boldsymbol{X}^2, \cdots, \boldsymbol{X}^N\}$ represents the $N$ training patterns. DistAl calculates the pairwise interpattern distances for the training set (using the chosen distance metric $d$) and stores them in the distance matrix $\mathscr{D}$. Each row of $\mathscr{D}$ is sorted in ascending order. Thus, row $k$ of $\mathscr{D}$ corresponds to the training pattern $\boldsymbol{X}^k$ and the elements $\mathscr{D}[k, i]$ correspond to the distance of $\boldsymbol{X}^k$ to the other training patterns. $\mathscr{D}[k, 0]$ is the distance to the closest pattern and $\mathscr{D}[k, N]$ is the distance to the farthest pattern from $\boldsymbol{X}^k$. Simultaneously, the attribute values of the training patterns are stored in $\mathscr{D}'$. $\mathscr{D}'$ is essentially the entire training set with $\mathscr{D}'[k, i]$ representing the $i$th attribute value of the $k$th training pattern. Each column (attribute) of $\mathscr{D}'$ is sorted in ascending order.

The key idea behind DistAl is to generate a single layer of hidden neurons each of which separates a subset of patterns in a training set using $\mathscr{D}$ (or $\mathscr{D}'$). Then, they are fully connected to $M$ output TLUs (one for each output class) in an output layer. The representation of the patterns at the hidden layer is linearly separable (34). Thus, an iterative perceptron learning rule can be used to train the output weights. However, the output weights can be directly set as follows: The weights between output and hidden neurons are chosen such that each hidden neuron overwhelms the effect of the hidden neurons generated later. If there are a total of $h$ hidden neurons (numbered 1, 2, . . ., $h$ from left to right) then the weight between the output neuron $j$ and the hidden neuron $i$ is set to $2^{h-i}$ if the hidden neuron $i$ excludes patterns belonging to class $j$ and zero otherwise.

Let $\boldsymbol{W}_l^h$ be the weights between the $l$th hidden neuron and inputs. Let $\boldsymbol{W}_m^o$ be the weights between the output neuron for class $m$ and hidden neurons, and $W_{ml}^o$ be the weight between the output neuron for class $m$ and the $l$th hidden neuron, respectively. Figure 1 summarizes the process of network construction.

### Use of Network in Classification

The outputs in the output layer are computed by the *winner-take-all* (WTA) strategy. The output neuron $m$ that has the highest net input produces 1 and all the other neurons produce 0s. The WTA strategy and the weight setting explained in the previous section guarantee 100% training accuracy for any finite noncontradictory set of training patterns.

Each test pattern is fed into the network and the outputs are computed by the WTA strategy. If there is one or more hidden neuron that produces 1 (i.e., there exists one or more hidden neuron that include the test pattern within their thresholds), the outputs are computed by the WTA strategy in the output layer. Otherwise (i.e., all hidden neurons produce 0s and all output neurons produce 0s as well), the distance between the test pattern and the thresholds of each hidden neuron is computed. The hidden neuron that has the minimum distance is chosen to produce 1. Then the outputs are computed again in the output layer to compare with the desired classification.

***Example.*** Although DistAl works on tasks with multicategory real-valued patterns, we illustrate its operation using the simple XOR problem. We assume the use of the Manhattan distance metric. There are four training patterns ($S = \{\boldsymbol{X}^1, \boldsymbol{X}^2, \boldsymbol{X}^3, \boldsymbol{X}^4\}$):

| | Input | | Class |
|---|---|---|---|
| $\boldsymbol{X}^1$: | 0 | 0 | A |
| $\boldsymbol{X}^2$: | 0 | 1 | B |
| $\boldsymbol{X}^3$: | 1 | 0 | B |
| $\boldsymbol{X}^4$: | 1 | 1 | A |

This yields the following distance matrix after sorted:

$$\mathscr{D} = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{pmatrix}$$

The first row of the matrix is the distance of $\boldsymbol{X}^1$, $\boldsymbol{X}^2$, $\boldsymbol{X}^3$, and $\boldsymbol{X}^4$ from pattern $\boldsymbol{X}^1$. The second row of the matrix is the distance of $\boldsymbol{X}^2$, $\boldsymbol{X}^1$, $\boldsymbol{X}^4$, and $\boldsymbol{X}^3$ from $\boldsymbol{X}^2$. The third row of the matrix is the distance of $\boldsymbol{X}^3$, $\boldsymbol{X}^1$, $\boldsymbol{X}^4$, and $\boldsymbol{X}^2$ from $\boldsymbol{X}^3$. The last row of the matrix is the distance of $\boldsymbol{X}^4$, $\boldsymbol{X}^2$, $\boldsymbol{X}^3$, and $\boldsymbol{X}^1$ from $\boldsymbol{X}^4$.

$\boldsymbol{X}^1$ excludes the maximum number of patterns from a single class (i.e., $S_k = \{\boldsymbol{X}^2, \boldsymbol{X}^3\}$, class = B). A hidden neuron is introduced for this cluster with $\boldsymbol{W}_1^h = [0\ 0]$, $\theta_{\text{low}} = \theta_{\text{high}} = 1$, $W_{B1}^o = 1$, $W_{A1}^o = 0$. $\boldsymbol{X}^2$ and $\boldsymbol{X}^3$ are now eliminated from further consideration (i.e., $S = S - S_k = \{\boldsymbol{X}^1, \boldsymbol{X}^4\}$). The remaining patterns ($S_k = \{\boldsymbol{X}^1, \boldsymbol{X}^4\}$, class = A) can be excluded by any pattern (say, $\boldsymbol{X}^1$ again) with another hidden neuron with $\boldsymbol{W}_2^h = [0\ 0]$, $\theta_{\text{low}} = 0$, $\theta_{\text{high}} = 2$, $W_{A2}^o = 1$, $W_{B2}^o = 0$, $W_{A1}^o = W_{A1}^o * 2 = 0$, $W_{B1}^o = W_{B1}^o * 2 = 2$. Now the algorithm stops since the entire training set is correctly classified (i.e., $S = S - S_k = \phi$). Figure 2 shows the network construction process.

### Structural Learning Using Feature-Subset Selection

In pattern classification tasks, the choice of features (or attributes) used to represent patterns affect the following:

*Learning Time.* The attributes used to describe the patterns implicitly determine the search space that needs to be explored by the learning algorithm. The larger the search space, the more time the learning algorithm needs for learning a sufficiently accurate classification function (7,39).

*Number of Examples Needed.* All other things being equal, the larger the number of attributes used to describe the patterns, the larger is the number of examples need to learn a classification function to a desired accuracy (7,39).

*Cost of Classification.* In many real-world pattern classification tasks (e.g., medical diagnosis), some of the attributes may be observable symptoms and others might require diagnostic tests. Different diagnostic tests might have different costs as well as risks associated with them.

Initialize the number of hidden neurons: $h = 0$;
**while** $S \neq \phi$ **do**

1. Double all existing weights (if any) between hidden and output neurons: $\mathbf{W}^o_m = \mathbf{W}^o_m * 2\ \forall m$
2. Increment the number of hidden neurons: $h = h + 1$
3. Interpattern distance based:
   Identify a row $k$ of $\mathscr{D}$ that excludes the largest subset of patterns in $S$ that belong to the same class $m$ as follows:
   (a) **For** each row $r = 1, \ldots, N$ **do**
      i. Let $i_r$ and $j_r$ be column indices (corresponding to row $r$) for the matrix $\mathscr{D}$ such that the patterns corresponding to the elements $\mathscr{D}[r, i_r]$, $\mathscr{D}[r, i_r + 1], \ldots, \mathscr{D}[r, j_r]$ all belong to the same class and also belong to $S$.
      ii. Let $c_r = j_r - i_r + 1$ (the number of patterns excluded).
   (b) Select $k$ to be the one for which the corresponding $c_k$ is the largest: $k = \arg\max_r c_r$
   (c) Let $S_k$ be the corresponding set of patterns that are excluded by pattern $\mathbf{X}^k$, $d^k_{\text{low}} = \mathscr{D}[k, i_k]$ (distance to the closest pattern of the cluster) and $d^k_{\text{high}} = \mathscr{D}[k, j_k]$ (distance to the farthest pattern of the cluster).
4. (a) Define a spherical threshold neuron with $\mathbf{W}^h = \mathbf{X}^k$, $\theta_{\text{low}} = d^k_{\text{low}}$, $\theta_{\text{high}} = d^k_{\text{high}}$.
   (b) $S = S - S_k$
5. Connect the new hidden neuron to output neurons: $W^o_{mh} = 1$; $W^o_{nh} = 0\ \forall n \neq m$

**end while**

**Figure 1.** Pseudo-code for DistAl.

This presents us with a *feature subset selection problem* in automated design of pattern classifiers. The feature subset selection problem refers the task of identifying and selecting a useful subset of attributes to be used to represent patterns from a larger set of attributes. Satisfactory solution of this problem is particularly critical if instance-based, nearest-neighbor, or similarity-based learning algorithms like DistAl are used to build the classifier. This is due to the fact that such classifiers rely on the use of interpattern distances that are intricately linked to the choice of features used to represent the patterns. Presence of irrelevant or misleading features (e.g., social security numbers in a medical diagnosis task) can skew the distance calculation and hence adversely affect the generalization performance of the resulting classifier.

A detailed discussion of feature subset selection is beyond the scope of this article. The interested reader is referred to Refs. 40 and 41 for discussion of a variety of alternative approaches to feature subset selection. Feature selection typically improves the performance of DistAl.

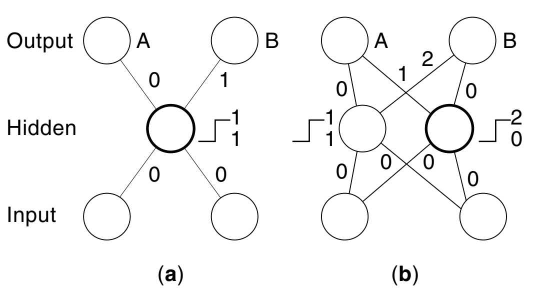


**Figure 2.** Process of network construction for the example in DistAl (a) after the first neuron is introduced and (b) after the second neuron is introduced (final network).

## SUMMARY

Constructive algorithms offer an attractive approach to the design of pattern classifiers. Some promising research directions include combining network pruning with network construction; incorporation of prior knowledge in network construction (42); and the use of constructive algorithms for cumulative multitask learning.

## ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation, grant IRI-9409580, and the John Deere Foundation.

## BIBLIOGRAPHY

1. U. Fayyad et al., *Advances in Knowledge Discovery and Data Mining,* Cambridge, MA: MIT Press, 1996.
2. V. Honavar, Machine learning: Principles and applications, in J. G. Webster (ed.), *Encyclopedia of Electrical and Electronics Engineering,* New York: Wiley, 1999.
3. J. Bradshaw, *Software Agents,* Cambridge, MA: MIT Press, 1997.
4. V. Honavar, Intelligent agents, in J. Williams and K. Sochats (eds.), *Encyclopedia of Information Technology,* New York: Marcel Dekker, 1998.
5. K. Balakrishnan and V. Honavar, Intelligent diagnosis systems. *Int. J. Intelligent Syst.,* in press.
6. R. Duda and P. Hart, *Pattern Classification and Scene Analysis,* New York: Wiley, 1973.
7. T. Mitchell, *Machine Learning,* New York: McGraw-Hill, 1997.
8. P. Langley, *Elements of Machine Learning,* Palo Alto, CA: Morgan Kaufmann, 1995.

9. V. Honavar, Toward learning systems that integrate multiple strategies and representations, in V. Honavar and L. Uhr (eds.), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration,* New York: Academic Press, 1994, pp. 615–644.
10. C.-H. Chen et al., Analysis of decision boundaries generated by constructive neural network learning algorithms, *Proc. WCNN '95,* Washington D.C., **1**, 1995, pp. 628–635.
11. S. Gallant, *Neural Network Learning and Expert Systems,* Cambridge, MA: MIT Press, 1993.
12. 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.
13. R. Parekh, J. Yang, and V. Honavar, MUPSTART—a constructive neural network learning algorithm for multicategory pattern classification. *Proc. IEEE/INNS Int. Conf. Neural Networks, ICNN '97,* 1997, pp. 1924–1929.
14. J. Yang, R. Parekh, and V. Honavar, MTILING—a constructive neural network learning algorithm for multicategory pattern classification. *Proc. World Congr. Neural Networks '96,* San Diego, 1996, pp. 182–187.
15. V. Honavar, Structural learning, in J. G. Webster (ed.), *Encyclopedia of Electrical and Electronics Engineering,* New York: Wiley, 1999.
16. D. Rumelhart, G. Hinton, and R. Williams, Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations into the Microstructure of Cognition,* Vol. 1 *(Foundations),* Cambridge, MA: MIT Press, 1986.
17. P. Werbos, *Beyond regression: new tools for prediction and analysis in behavioral sciences,* PhD thesis, Harvard University, Cambridge, MA, 1974.
18. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.,* **65**: 386–408, 1958.
19. N. Nilsson, *The Mathematical Foundations of Learning Machines,* New York: McGraw-Hill, 1965.
20. W. Krauth and M. Mézard, Learning algorithms with optimal stability in neural networks. *J. Phys. A: Math. Gen.,* **20**: L745–L752, 1987.
21. J. Anlauf and M. Biehl, Properties of an adaptive perceptron algorithm, in *Parallel Processing in Neural Systems and Computers,* 1990, pp. 153–156.
22. M. Frean, *Small nets and short paths: optimizing neural computation.* PhD thesis, Center for Cognitive Science, Edinburgh University, UK, 1990.
23. H. Poulard, Barycentric correction procedure: a fast method of learning threshold units. In *Proc. WCNN '95,* Washington DC, **1**: 1995, pp. 710–713.
24. B. Raffin and M. Gordon, Learning and generalization with minimerror, a temperature-dependent learning algorith, *Neural Computation,* **7**: 1206–1224, 1995.
25. R. Reed, Pruning algorithms—a survey, *IEEE Trans. Neural Networks,* **NN-4**: 740–747, 1993.
26. R. Parekh, J. Yang, and V. Honavar, Pruning strategies for constructive neural network learning algorithms, in *Proc. IEEE/INNS Int. Conf. Neural Networks, ICNN '97,* 1997, pp. 1960–1965.
27. V. Honavar, *Generative learning structures and processes for generalized connectionist networks,* PhD thesis, University of Wisconsin, Madison, WI, 1990.
28. V. Honavar and L. Uhr, Generative learning structures for generalized connectionist networks, *Inf. Sci.,* **70** (1–2): 75–108, 1993.
29. J. Nadal, Study of a growth algorithm for a feedforward network, *Int. J. Neural Syst.,* **1** (1): 55–59, 1989.
30. S. Gallant, Perceptron based learning algorithms, *IEEE Trans. Neural Networks,* **1**: 179–191, 1990.
31. M. Mézard and J. Nadal, Learning feed-forward networks: The tilting algorithm, *J. Phys. A: Math. Gen.,* **22**: 2191–2203, 1989.
32. M. Frean, The upstart algorithm: A method for constructing and training feedforward neural networks, *Neural Computation,* **2**: 198–209, 1990.
33. N. Burgess, A constructive algorithm that converges for real-valued input patterns, *Int. J. Neural Syst.,* **5** (1): 59–66, 1994.
34. M. Marchand, M. Golea, and P. Rujan, A convergence theorem for sequential learning in two-layer perceptrons, *Europhys. Lett.,* **11** (6): 487–492, 1990.
35. E. Diday, Recent progress in distance and similarity measures in pattern recognition, in *Proc. 2nd Int. Joint Conf. Pattern Recognition,* 1974, pp. 534–539.
36. G. Salton and M. McGill, *Introduction to Modern Information Retrieval,* New York: McGraw-Hill, 1983.
37. B. Batchelor, *Pattern Recognition: Ideas in Practice,* New York: Plenum, 1978.
38. D. Wilson and T. Martinez, Improved heterogeneous distance functions, *J. Artif. Intell. Res.,* **6**: 1–34, 1997.
39. B. Natarajan, *Machine Learning: A Theoretical Approach,* San Mateo, CA: Morgan Kauffman, 1991.
40. J. Yang and V. Honavar, Feature subset selection using a genetic algorithm, *IEEE Expert (Special Issue on Feature Transformation and Subset Selection),* **13** (2), 44–49, 1998.
41. J. Yang and V. Honavar, Feature subset selection using a genetic algorithm, in *Feature Extraction, Construction and Selection—A Data Mining Perspective,* New York: Kluwer Academic, in press.
42. R. Parekh and V. Honavar, Constructive theory refinement using knowledge based neural networks, in *Proc. INNS/IEEE Joint Conference on Neural Networks,* IJCNN 98, (in press).

VASANT HONAVAR
JIHOON YANG
RAJESH PAREKH
Iowa State University

**CONSUMER PRODUCT DESIGN.** See DESIGNING CONSUMER PRODUCTS FOR EASE OF USE.

**CONTACT ELECTRIFICATION.** See TRIBOELECTRICITY.