DistAI: An Inter-pattern Distance-based Constructive Learning Algorithm

TR #97-05

Jihoon Yang, Rajesh Parekh and Vasant Honavar

February 10, 1997

ACM Computing Classification System Categories (1991):
I.2.6 [*Artificial Intelligence*] Learning — concept learning, connectionism and neural nets.

Keywords:
pattern classification systems, multi-layer neural networks, constructive learning algorithms, inter-pattern distance metrics.

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

DistAI: An Inter-pattern Distance-based Constructive Learning Algorithm

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

Abstract

Multi-layer networks of threshold logic units offer an attractive framework for the design of pattern classification systems. A new constructive neural network learning algorithm (DistAI) based on inter-pattern distance is introduced. DistAI uses spherical threshold neurons in a hidden layer to find a cluster of patterns to be covered (or classified) by each hidden neuron. It does not depend on an iterative, expensive and time-consuming perceptron training algorithm to find the weight settings for the neurons in the network, and thus extremely fast even for large data sets. The experimental results (in terms of generalization capability and network size) of DistAI on a number of benchmark classification problems show reasonable performance compared to other learning algorithms despite its simplicity and fast learning time. Therefore, DistAI is a good candidate to various tasks that involve very large data sets (such as largescale datamining and knowledge acquisition) or that require reasonably accurate classifiers to be learned in almost real time or that use neural network learning as the inner loop of a more complex optimization process in hybrid learning systems.

1 Introduction

Multi-layer networks of threshold logic units (TLU) or multi-layer perceptrons (MLP) [Chen *et al.*, 1995; Gallant, 1993; Parekh *et al.*, 1996; Parekh *et al.*, 1995] 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

*This research was partially supported by the National Science Foundation (through grants IRI-9409580 and IRI-9643299) and the John Deere Foundation.

trees [Gallant, 1993]; and simpler digital hardware implementations than their continuous counterparts such as sigmoid neurons used in back-propagation networks [Rumelhart *et al.*, 1986].

A TLU implements an $(N-1)$-dimensional hyperplane which partitions 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 [Gallant, 1993; Frean, 1990a; Poulard, 1995; Raffin & Gordon, 1995; Anlauf & Biehl, 1990; Krauth & Mézard, 1987]. However, when the given set of patterns is not linearly separable, a multi-layer 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 multi-layer neural networks for pattern classification:

- *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 (typically error gradient guided) search for a suitable setting of numerical parameters, weights in a weight space defined by the choice of the network topology.

- *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 useless neurons and connections are used in conjunction with the network construction mechanisms.

Some of the motivations for study of constructive learning algorithms are [Honavar, 1990; Honavar & Uhr, 1993; Parekh *et al.*, 1995; Parekh *et al.*, 1996]:

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

- *Complexity of the network should match the intrinsic complexity of the classification task:* It is desirable that a learning algorithm construct networks whose complexity (as measured in terms of relevant criteria such as number of nodes, number of links, connectivity, etc.) is commensurate with the intrinsic complexity of the classification task

(implicitly specified by the training data). Smaller networks yield efficient hardware implementations. And everything else being equal, the more compact the network, the more likely it is that it exhibits better generalization properties. Constructive algorithms can potentially discover near-minimal networks for correct classification of a given data set.

- *Estimation of expected case complexity of pattern classification tasks:* Many pattern classification tasks are known to be computationally hard. However, little is known about the *expected* case complexity of classification tasks that are encountered, and successfully solved, by living systems - primarily because it is difficult to mathematically characterize the statistical distribution of such problem instances. Constructive algorithms, if successful, can provide useful empirical estimates of expected case complexity of real-world pattern classification tasks.

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

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

A number of constructive algorithms that incrementally construct networks of threshold neurons for 2-category pattern classification tasks have been proposed in the literature. These include the *tower*, *pyramid* [Gallant, 1990], *tiling* [Mézard & Nadal, 1989], *upstart* [Frean, 1990b], *perceptron cascade* [Burgess, 1994], and *sequential* [Marchand *et al.*, 1990]. More recently, [Parekh *et al.*, 1995; Parekh *et al.*, 1996] have proposed provably convergent multi-category versions of constructive algorithms for both discrete as well as real-valued pattern classification. With the exception of the sequential learning algorithm, most of 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:

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

- Training the added threshold neuron(s) using some variant of the perceptron training algorithm (e.g., the pocket algorithm [Gallant, 1993]) 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. Because of the iterative nature of the perceptron training algorithm, this is a time consuming process. This often makes the use of such algorithms impractical for very large data sets (e.g., in largescale datamining and knowledge acquisition tasks), especially in applications where reasonably accurate classifiers have to learned in almost real time. Similarly, hybrid learning systems that use neural network learning as the inner

loop of a more complex optimization process (e.g., feature subset selection using a genetic algorithm where evaluation of fitness of a solution requires training a neural network based on a subset of input features represented by the solution and evaluating its classification accuracy [Yang & Honavar, 1997]) call for a fast neural network training algorithm. This paper presents a new constructive learning algorithm (DistAl) which is designed with such applications in mind.

The rest of the paper is organized as follows: Section 2 describes DistAl. Section 3 presents the results of experiments designed to evaluate the performance of neural networks trained using DistAl on some benchmark classification problems. Section 4 concludes with a summary and discussion of some directions for future research.

2 DistAl: A New Constructive Learning Algorithm

DistAl differs from most other constructive learning algorithms mentioned above in two respects:

- It uses a variant of TLUs (spherical threshold units) as hidden neurons. (For motivations and explanations about spherical threshold units, see [Langley, 1995]). A spherical threshold neuron i has associated with it a weight vector $\mathbf{W}_i$, two thresholds — $\theta_{i,low}$ and $\theta_{i,high}$, and a suitably defined distance metric d. It computes the distance $d(\mathbf{W}_i, \mathbf{X}^p)$ between a given input pattern $\mathbf{X}^p$ and $\mathbf{W}_i$. The corresponding output $o_i^p = 1$ if $\theta_{i,low} \leq d(\mathbf{W}_i, \mathbf{X}^p) \leq \theta_{i,high}$ and 0 otherwise.
- DistAl does not use an iterative algorithm for finding the weights to set the weights and the thresholds. Instead, it computes the inter-pattern distances once between each pair of patterns in the training set and determines the weight values for hidden neurons by greedy strategy (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 section 2.4 for details).

The use of one-time inter-pattern distance calculation instead of (usually) iterative, expensive and time-consuming perceptron training procedure make 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 (see section 2.6 for details). This makes DistAl particularly well-suited for largescale datamining tasks.

2.1 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). Different distance metrics represent different notions of *distance* in the pattern

space. The number and distribution of the clusters that result is a function of the distribution of the patterns as well as the clustering strategy used. Since 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 [Duda & Hart, 1973; Salton & McGill, 1983].

Let $\mathbf{X}^p = [X_1^p, \cdots, X_n^p]$ and $\mathbf{X}^q = [X_1^p, \cdots, X_n^q]$ be two pattern vectors. Let max_i and min_i be the maximum and the minimum values of the ith attribute of patterns in a data set, respectively. Then the distance between $\mathbf{X}^p$ and $\mathbf{X}^q$ is defined as follows in each distance metric:

1. Euclidean:
$$d(\mathbf{X}^p, \mathbf{X}^q) = \sqrt{\sum_{i=1}^{n}(X_i^p - X_i^q)^2}$$

2. Manhattan:
$$d(\mathbf{X}^p, \mathbf{X}^q) = \sum_{i=1}^{n} |X_i^p - X_i^q|$$

3. Maximum Value:
$$d(\mathbf{X}^p, \mathbf{X}^q) = argmax_i |X_i^p - X_i^q|$$

4. Normalized Euclidean:
$$d(\mathbf{X}^p, \mathbf{X}^q) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\frac{X_i^p - X_i^q}{max_i - min_i})^2}$$

5. Normalized Manhattan:
$$d(\mathbf{X}^p, \mathbf{X}^q) = \frac{1}{n}\sum_{i=1}^{n}\frac{|X_i^p - X_i^q|}{max_i - min_i}$$

6. Normalized Maximum Value:
$$d(\mathbf{X}^p, \mathbf{X}^q) = argmax_i \frac{|X_i^p - X_i^q|}{max_i - min_i}$$

7. Dice coefficient:
$$d(\mathbf{X}^p, \mathbf{X}^q) = 1 - \frac{2\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2}$$

8. Cosine coefficient:
$$d(\mathbf{X}^p, \mathbf{X}^q) = 1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sqrt{\sum_{i=1}^{n}(X_i^p)^2 \cdot \sum_{i=1}^{n}(X_i^q)^2}}$$

9. Jaccard coefficient:
$$d(\mathbf{X}^p, \mathbf{X}^q) = 1 - \frac{\sum_{i=1}^{n} X_i^p X_i^q}{\sum_{i=1}^{n}(X_i^p)^2 + \sum_{i=1}^{n}(X_i^q)^2 - \sum_{i=1}^{n} X_i^p X_i^q}$$

2.2 Hidden Neurons

As mentioned above, DistAI introduces a hidden neuron to represent a cluster of training patterns that belong to the same class. The boundaries of a cluster are determined based on the inter-pattern distances and implemented by thresholds. DistAI uses a variant of TLUs (spherical threshold units) for this job: A hidden neuron is a concentric spherical threshold neuron representing a cluster of patterns by its upper and lower thresholds. It outputs 1 for the patterns whose distances from the center of the cluster are between the upper and the lower thresholds and 0 for all the other patterns. A hidden neuron finds a cluster using each distance metric in section 2.1 by greedy strategy (i.e., it finds one that corresponds to the maximal subset of patterns from the set of the patterns that are not included in any other clusters represented by hidden neurons generated so far).

In addition that various distance metrics were introduced in section 2.1 to explore the pattern space and to generate clusters for hidden neurons without any a-priori knowledge on the distribution of patterns, there might be some properties in training patterns that can be exploited. For example, the values of a particular attribute of all (or a subset of) patterns belonging to a class can be in an interval exclusively (i.e., the values of the attribute of patterns belonging to other classes are always less than the lower bound or greater than the upper bound of the interval), and thus the patterns of a class can be classified by considering that attribute only instead of computing the inter-pattern distances. Therefore, this *attribute-based* approach is also considered with the greedy strategy as in the distance-based approach (i.e., the maximal subset of patters belonging to a class is determined by sorting and scanning the values of each attribute of the training patterns). In attribute-based approach, a hidden neuron outputs 1 if the value of an attribute (it remembers) is within its thresholds, and 0 otherwise.

2.3 Output Neurons

DistAI uses general TLUs as output neurons. It computes the weighted sum of the outputs of hidden neurons. The value of output neurons are determined by the *winner-take-all* strategy: only an output neuron that has the largest net input outputs 1, and all the other output neurons output 0's; if there is a tie, all output neurons output 0's.

2.4 Network Construction

First of all, the distance between each pair of input patterns is calculated, sorted and stored in the distance matrix $\mathcal{D}_k, k \in \{1, \cdots, 9\}$ using the corresponding distance metrics. For attribute-based approach, the values of each attribute of the input patterns are sorted and stored in a matrix (say, $\mathcal{D}_{10}$ for notational simplicity).

DistAI maintains a single hidden layer throughout the learning phase. Starting with the entire training patterns and no hidden neurons, it keeps choosing a subset of patterns in the same class from the training set (using $\mathcal{D}_k, k \in \{1, \cdots, 10\}$) by introducing a hidden neuron until every pattern is chosen. In distance-based approach, it checks the maximum number of consecutive patterns of a class from each pattern in $\mathcal{D}_k, k \in \{1, \cdots, 9\}$ (i.e., the number of patterns of the same class in a cluster centered at each pattern). In attribute-

based approach, it checks the maximum number of consecutive patterns of the same class sorted by the values of each attribute in $\mathcal{D}_{10}$ (i.e., the number of patterns in intervals of patterns of the same class for each attribute).

In distance-based approach, the weights between the new hidden neuron and inputs are set by the pattern which is the center of the cluster. In this case, θ_{low} is the distance to the nearest pattern and θ_{high} is the distance to the farthest pattern in the cluster from the center. In attribute-based approach, the attribute used for determining the maximal subset is remembered. The weights between the new hidden neuron and inputs are ignored (because they are not used when the output is computed). In this case, θ_{low} is the smallest attribute value and θ_{high} is the largest attribute value of the patterns in the maximal subset.

The patterns belonging to the maximal subset is now eliminated from the training set and the process is repeated until all the patterns are eliminated. After all patterns are eliminated, the representation (of hidden neurons) becomes linearly separable [Marchand *et al.*, 1990]. As the weights between hidden and input neurons are determined from the distance matrices (not by the perceptron algorithm), following weight setting can be used between output and hidden neurons without applying the perceptron algorithm as well: The weights between output and hidden neurons are properly chosen for each hidden neuron to overwhelm the effect of other hidden neurons generated later. One simple strategy is to give $2^{s-1}, 2^{s-2}, \cdots, 2^0$ as the weights between the right output neurons (i.e., correct classification) and hidden neurons $1, 2, \cdots, s$, respectively [Marchand *et al.*, 1990], and 0's to all the other connections.

In terms of the speed of DistAI, because patterns that are classified by a new hidden neuron once are eliminated from the training set and not considered further and no perceptron training algorithm is applied, DistAI is significantly fast.

2.5 Pseudo-code

Let $\mathcal{S}$ be a set of training patterns and i denote the number of hidden neurons generated. Let $\mathbf{W}_i^h$ be the weights between the ith hidden neuron and inputs, and let W_{ji}^o be the weight between jth output neuron and ith hidden neuron.

1. Compute a distance matrix $\mathcal{D}_k(\mathbf{X}_i, \mathbf{X}_j)\ \forall \mathbf{X}_i, \mathbf{X}_j \in \mathcal{S}, k \in \{1, \cdots, 9\}$ or the attribute-based matrix $\mathcal{D}_{10}$, and sort it by an ascending order;
2. $i = 0$;
3. while $(\mathcal{S} \neq \phi)$ do begin
4. Double all existing weights between output and hidden neurons (initially, there is no hidden neuron);
5. Introduce a new hidden neuron i $(i = i + 1)$;
6. Select a maximal subset $\mathcal{S}'$ of patterns (belonging to a class, say, j) from $\mathcal{S}$ using $\mathcal{D}_k,\ k \in \{1, \cdots, 10\}$
7. if ($\mathcal{S}'$ is chosen by distance-based approach [$\mathbf{X}^p$: center, $\mathbf{X}^f$: first, $\mathbf{X}^l$: last patterns of the cluster in $\mathcal{D}_k$, respectively]) then

8. $\mathbf{W}_i^h \leftarrow \mathbf{X}^p$; $\theta_{low} = \mathcal{D}_k(\mathbf{X}^p, \mathbf{X}^f)$; $\theta_{high} = \mathcal{D}_k(\mathbf{X}^p, \mathbf{X}^l)$;
else [$\mathcal{S}'$ is chosen using $\mathcal{D}_{10}$ with attribute r, smallest value: s, largest value: l]

9. $\theta_{low} = s$; $\theta_{high} = l$; Store attribute index r;

10. $W_{ji}^o \leftarrow 1$; $W_{mi}^o \leftarrow 0, \forall m \neq j$;

11. $\mathcal{S} = \mathcal{S} - \mathcal{S}'$;

end

2.6 Complexity

It is clear that there exists at least one pattern that can be eliminated by a hidden neuron at any time with the distance-based approach unless the training set is contradictory (i.e., there exist identical patterns with different classifications). Note that this is not generally true for attribute-based approach because there can be many patterns belonging to different classes that have the same attribute values. Thus, following complexity analysis and convergence proof apply to distance-based approach only.

Let N_{pat} be the number of training patterns and N_{att} be the number of attributes in a data set, respectively. Let N_{out} and N_{hidden} be the number of output and hidden neurons, respectively. Assume $N_{pat} > N_{att}$ and $N_{pat} \gg max[N_{out}, N_{hidden}]$.

2.6.1 Time Complexity

Step 1 takes $\mathcal{O}(max[N_{pat}^2 \cdot N_{att}, N_{pat}^2 \cdot \log N_{pat}])$. Steps 2,5,7 and 9 take $\mathcal{O}(1)$. Step 4 takes $\mathcal{O}(N_{out} \cdot N_{hidden})$. Step 6 takes $\mathcal{O}(N_{pat}^2)$. Step 8 takes $\mathcal{O}(N_{att})$. Step 10 takes $\mathcal{O}(N_{out})$. Step 11 takes $\mathcal{O}(N_{pat})$. Thus, step 3 (worst-case) takes $\mathcal{O}(N_{pat}^3)$. Therefore, worst-case time complexity is $\mathcal{O}(N_{pat}^3)$. (Note that DistAI runs much faster than the worst-case time complexity because it keeps eliminating a subset of elements from the original training set).

2.6.2 Space Complexity

The space requirement for the input patterns and their targets is $\mathcal{O}(N_{pat} \cdot [N_{att} + N_{out}])$. The weights require $\mathcal{O}(N_{out} \cdot N_{hidden} + N_{hidden} \cdot N_{in})$. The distance matrix requires $\mathcal{O}(N_{pat}^2)$. Thus, the total space complexity is $\mathcal{O}(N_{pat}^2)$.

2.7 Example

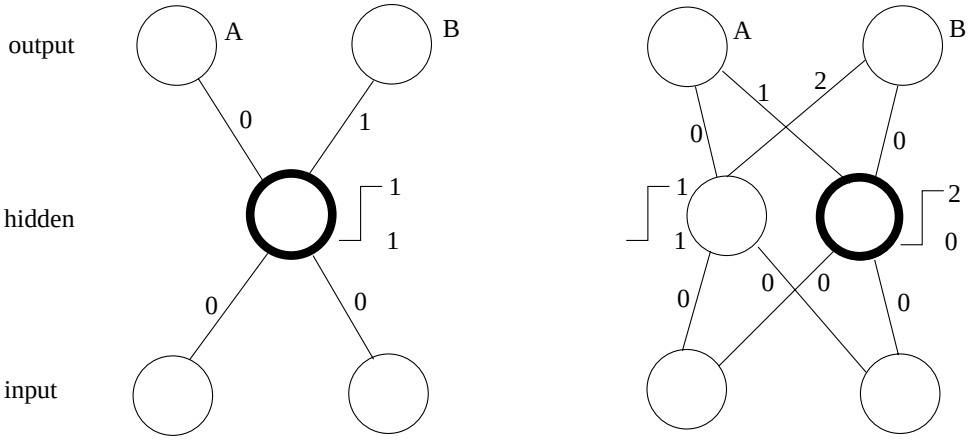
Consider the XOR problem assuming Manhattan distance metric is used ($\mathcal{S} = \{1, 2, 3, 4\}$):

input			class
pattern 1:	0	0	A
pattern 2:	0	1	B
pattern 3:	1	0	B
pattern 4:	1	1	A

Then, the distance matrix will be like the following after sorted:

$$\begin{pmatrix} 0\ 1\ 1\ 2 \\ 0\ 1\ 1\ 2 \\ 0\ 1\ 1\ 2 \\ 0\ 1\ 1\ 2 \end{pmatrix} \begin{array}{l} \rightarrow \text{from pattern 1 to patterns } (1,2,3,4) \\ \rightarrow \text{from pattern 2 to patterns } (2,1,4,3) \\ \rightarrow \text{from pattern 3 to patterns } (3,1,4,2) \\ \rightarrow \text{from pattern 4 to patterns } (4,2,3,1) \end{array}$$

Thus, the maximum consecutive patterns of a class is from pattern 1 to patterns 2 and 3 (i.e., $\mathcal{S}' = \{2,3\}$), and a hidden neuron is introduced for this cluster with $\theta_{low} = \theta_{high} = 1$. Then, patterns 2 and 3 are eliminated from further consideration, which leaves pattern 1 and 4 (i.e., $\mathcal{S} = \{1,4\}$) and that can be excluded from any pattern (say, pattern 1 again, $\mathcal{S}' = \{1,4\}$) with another hidden neuron with $\theta_{low} = 0, \theta_{high} = 2$. So, $\mathcal{S} = \phi$, and the algorithm stops. Figure 1 shows this process of network construction.



(a) after the first neuron is introduced

(b) after the second neuron is introduced (final network)

Figure 1: Process of Network Construction for the Example in DistAl

2.8 Convergence Proof

From the algorithmic description, we clearly have the following theorem similar to the one in [Marchand *et al.*, 1990].

Theorem:
DistAl guarantees to converge to 100% training accuracy with a finite number of hidden neurons for a data set with a finite number of non-contradictory patterns.
(Proof)
Because hidden neurons are generated by the sequence of eliminating a subset of patterns from the training set (i.e., clustering them by greedy strategy), the internal representation of the hidden layer for a pattern $\mathbf{X}^p$ (member of the k-th cluster) has the form

$$\mathbf{H}^p = (0, 0, \cdots, 0, 1, *, \cdots, *)$$

(it has 0's in the first $k-1$ hidden neurons, 1 in the k-th hidden neuron and either 0 or 1 in the remaining hidden neurons) after a network is constructed consuming h hidden neurons.

Consider the following weight settings:

$$W_{ji}^{o} = \begin{cases} 2^{h-i} \text{ if } j \text{ is the right class of hidden neuron } i \\ 0 \text{ otherwise} \end{cases}$$

Then $\mathbf{X}^p$ is correctly classified in the output layer by the winner-take-all strategy because the cluster (represented by a hidden neuron) to which $\mathbf{X}^p$ belongs gives the largest net input which wins over other net inputs (computed by hidden neurons generated later) to output neurons.

Because there will be a finite number of clusters for a finite number of patterns in a data set, a finite number of hidden neurons will be generated and 100% training accuracy is obtained. □

3 Experiments

3.1 Data sets

A wide range of real-world as well as artificial data sets were used to see the performance of DistAI. The parity, random function and two spirals problems were chosen as artificial data sets. The real-world data sets were obtained from the machine learning data repository at the University of California at Irvine [Murphy & Aha, 1994]. Table 1 summarizes the characteristics of the data sets selected for our experiments.

3.2 Experimental Results

DistAI is deterministic in the sense that its behavior is always identical for a given training set while most other constructive learning algorithms are non-deterministic because their behavior is not always identical in different runs with the same training set and even with the same learning parameters. DistAI was run once for each distance metric to compare the performance in terms of the generalization accuracy and the network size (i.e., number of hidden neurons generated). The generalization accuracy is computed as follows: If there is no tie in output neurons for a test pattern, DistAI checks the correctness by comparing the desired outputs with computed outputs by winner-take-all strategy; if there is a tie (i.e., every hidden neuron outputs 0), it considers the distance from the test pattern to the sphere (or thresholds) of each cluster and picks a hidden neuron which has the minimum distance and outputs 1 for that hidden neuron. Then it computes the outputs (in output layer) and compares them with the desired classification.

All the artificial data sets have only training sets and thus generalization accuracy is not considered. Parity and random function data sets contain bipolar patterns, which make DistAI behave similarly for different distance metrics in section 2.1 and thus only Manhattan distance metric is used for those data sets.

The results are shown in table 2. A '*' indicates failure to achieve 100% training accuracy with the limit of 100 hidden neurons and/or convergence with 0% generalization accuracy.

Dataset	train	test	att	type	class
balance scale weight & distance (**balance**)	416	209	4	int	3
glass identification (**glass**)	142	72	9	real	6
ionosphere structure (**ionosphere**)	234	117	34	real, int	2
BUPA liver disorders (**liver**)	230	115	6	real, int	2
7-bit parity (**p7**)	128	0	7	bipolar	2
8-bit parity (**p8**)	256	0	8	bipolar	2
9-bit parity (**p9**)	512	0	9	bipolar	2
pima indians diabetes (**pima**)	512	256	8	real, int	2
5-bit random function (**r5**)	32	0	5	bipolar	3
image segmentation (**segmentation**)	210	2100	19	real, int	7
two spirals (**spirals**)	192	0	2	real	2
tic-tac-toe endgame (**tic-tac-toe**)	638	320	9	int	2
vowel recognition (**vowel**)	528	462	10	real	11
wisconsin diagnostic breast cancer (**wdbc**)	380	189	30	real	2
wine recognition (**wine**)	120	58	13	real, int	3

Table 1: Datasets used in experiments. train is the number of training patterns, test is the number of test patterns, att is the number of input attributes, type is the attribute type and class is the number of classes.

As pointed earlier, the distance-based approach is guaranteed to converge to 100% training accuracy but the attribute-based approach is not. For example, the attribute values in **tic-tac-toe** are the same in patterns of different classifications in majority of the data set, and the algorithm did not converge by attribute-based approach. It is also possible DistAI does not converge to 100% training accuracy given a limited number of hidden neurons. If a data set has a number of training patterns and its distribution is not so appropriate to cluster by the distance or attributed-based methods, it may not converge after consuming all the hidden neurons.

Because of the fast speed of DistAI, the results were obtained instantly even with all the distance metrics. As we can see from table 2, there is no well-defined rule of choosing the best distance metric for each data set. That is because the performance (or clustering) depends on the distribution of a data set. In other words, a specific distance metric might be appropriate for a data set while it might not for other data sets.

It is generally impossible to do a perfect, thorough and fair comparison between various learning algorithms because each algorithm has its own *optimal* parameter settings which is usually unknown and not feasible to obtain within a reasonable amount of time. We compare the results of DistAI on several data sets with the results obtained by a few learning algorithms that are reported in the literature.

3.2.1 Artificial Data Sets

- Parity

The number of hidden neurons generated in DistAI is comparable to that of *perceptron cascade* [Burgess, 1994] (3,4,4 for 7,8,9-bit parity problems) which they claim to be as efficient as any of existing constructive algorithms, and, of course, DistAI converged very fast.

- spirals
  It is reported in [Burgess, 1994] that *perceptron cascade* generated 35 or 18 (depending on the implementation), *cascade correlation* [Fahlman & Lebiere, 1990] generated 15, and *upstart* [Frean, 1990b] generated 91 hidden neurons. DistAI needed only 10 hidden neurons with Euclidean distance metric.

3.2.2 Real-world Data Sets

The generalizations of DistAI on real-world data sets are either comparable to or slightly worse than the best results reported so far in [Murphy & Aha, 1994]. (Note that comparing the results of other learning algorithms in the literature with those of DistAI (obtained without any optimization) is not fair because there is no way of knowing exactly how the results are obtained such as the algorithms used, various parameter settings, whether the algorithm was optimized or finetuned for a given problem, how the training and test sets are divided, etc.).

- **ionosphere**
  Networks with hidden neurons in the range of [0,15] using the backpropagation learning algorithm gave 96% accuracy after training with 200 patterns [Sigillito *et al.*, 1989], and IB3 gave 96.7% accuracy [Aha & Kibler, 1989]. DistAI gave a comparable performance of 94% with Jaccard distance metric.

- **pima**
  The ADAP algorithm makes a real-valued prediction between 0 and 1, and this was transformed into a binary decision using a cutoff of 0.448. It used 576 training patterns and gave 76% accuracy for the remaining 192 patterns [Smith *et al.*, 1988]. DistAI gave 68% accuracy with attribute-based method.

- **tic-tac-toe**
  Results of 6 algorithms were reported — NewID: 84.0%, CN2: 98.1%, MBRtalk: 88.4%, IB1: 98.1%, IB3: 82.0%, IB3-CI: 99.1% [Aha, 1991]. The best result of DistAI was 82.5% using Cosine distance metric. Originally, all the attribute values were nominal in **tic-tac-toe** and those were converted into integer values by an arbitrary way. This arbitrary and ad hoc data conversion can skew the data set and cause it to lose its original characteristics, and make the performance of DistAI (which considers the inter-pattern distance or the relative magnitude of attribute values) very poor with some distance metrics.

- **vowel**
  The results of a number of algorithms are given in [Robinson, 1989]. The accuracies range between 33% and 56%. DistAI gave 37% using maximum value distance metric. The performance might be improved using larger training set including more training patterns from more speakers.

- **wdbc**
  The best performance is 97.5% using repeated 10-fold cross-validations [Mangasarian *et al.*, 1995] that is much more computationally intensive than DistAI. DistAI gave a comparable performance of 92.1% with maximum value distance metric.

- **wine**
  The results of algorithms are in [Aeberhard *et al.*, 1992] — RDA:100%, QDA:99.4%, LDA: 98.9%, 1NN: 96.1%. DistAI gave a comparable performance of 94.8% with normalized Manhattan distance metric.

4 Summary and Discussion

A significantly fast, inter-pattern distance-based constructive learning algorithm, DistAI, is introduced and its performance on a number of artificial and real-world data sets is shown. Despite its simplicity, DistAI yielded a reasonable performance on almost all data sets considered (and excellent performance on difficult tasks such as the parity and two spirals problems) compared with other learning algorithms. Due to its significantly fast speed, it is well-suited to many real-world applications involving huge amount of data and/or requesting real-time response such as largescale datamining and knowledge acquisition tasks and hybrid learning systems that use neural network learning as the inner loop of a more complex optimization process like feature subset selection using a genetic algorithm [Yang & Honavar, 1997].

Theoretical analysis of DistAI on generalization capability is currently being explored and will strengthen the algorithm in addition to its fast speed. Also, performance comparison of DistAI with other various constructive learning algorithms (based on extensive experiments with a number of data sets) is in progress.

References

Aeberhard, S., Coomans, D., & de Vel, O. (1992). *Comparison of Classifiers in High Dimensional Settings.* Tech. rept. TR92-02. Department of Computer Science and Department of Mathematics and Statistics, James Cook University of North Queensland.

Aha, D. (1991). Incremental Constructive Induction: An Instance-based Approach. *Pages 117–121 of: Proceedings of the Eighth International Workshop on Machine Learning.* Evanston, IL: Morgan Kaufmann.

Aha, D., & Kibler, D. (1989). Noise-tolerant Instance-based Learning Algorithms. *Pages 794–799 of: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Detroit, MI: Morgan Kaufmann.

Anlauf, J. K., & Biehl, M. (1990). Properties of an Adaptive Perceptron Algorithm. *Pages 153–156 of: Parallel Processing in Neural Systems and Computers.*

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

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

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

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

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

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

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

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

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

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

Krauth, W., & Mézard, M. (1987). Learning Algorithms with Optimal Stability in Neural Networks. *J. Phys. A: Math. Gen.*, **20**, L745–L752.

Langley, P. (1995). *Elements of Machine Learning.* Palo Alto, CA: Morgan Kaufmann.

Mangasarian, O., Street, W., & Wolberg, W. (1995). Breast Cancer Diagnosis and Prognosis via Linear Programming. *Operations Research*, **43**(4), 570–577.

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

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

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

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

Parekh, R., Yang, J., & Honavar, V. (1996). *Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification.* Tech. rept. TR 96-14. Department of Computer Science, Iowa State University.

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

Raffin, B., & Gordon, M. G. (1995). Learning and Generalization with Minimerror, A Temperature-Dependent Learning Algorithm. *Neural Computation*, **7**, 1206–1224.

Robinson, A. (1989). *Dynamic Error Propagation Networks.* Ph.D. thesis, Cambridge University Engineering Department, UK.

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

Salton, G., & McGill, M. (1983). *Introduction to Modern Information Retrieval.* New York: McGraw Hill.

Sigillito, V., Wing, S., Hutton, L., & Baker, K. (1989). Classification of Radar Returns from the Ionosphere Using Neural Networks. *Johns Hopkins APL Technical Digest*, **10**, 262–266.

Smith, J., Everhart, J., Dickson, W., Knowler, W., & Johannes, R. (1988). Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. *Pages 261–265 of: Proceedings of the Symposium on Computer Applications and Medical Care.* IEEE Computer Society Press.

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

Dataset	Eucl		Manh		Max val		N. Eucl		N. Manh		N. Max val		Dice		Cosine		Jaccard		Attr-based		Reported Best
	A	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	*A*	*H*	
balance	83.25	15	85.17	16	77.51	89	83.25	15	85.17	16	77.51	89	76.56	15	**89.00**	12	76.56	15	*	*	-
glass	43.06	19	43.06	20	41.67	18	38.89	18	38.89	19	38.89	22	44.44	21	43.06	23	44.44	21	**54.17**	27	-
ionosphere	73.50	13	86.32	12	90.60	11	*	*	*	*	76.92	10	83.76	10	81.20	9	**94.02**	10	87.18	14	97
liver	62.61	21	57.39	20	**71.30**	23	48.70	20	50.43	20	63.48	24	58.26	21	52.17	20	60.87	20	58.26	53	-
p7	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p8	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
p9	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
pima	63.28	37	63.67	33	66.41	54	65.66	33	58.98	32	64.84	40	57.42	37	60.55	41	58.59	36	**67.97**	72	76
r5	-	-	-	14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
segmentation	63.86	23	65.67	22	61.67	31	*	*	*	*	66.14	27	69.43	23	55.33	32	69.52	23	**90.52**	18	-
spirals	-	10	-	15	-	16	-	11	-	17	-	17	-	22	-	36	-	23	-	39	-
tic-tac-toe	51.88	20	32.19	18	*	*	51.88	20	29.69	18	*	*	71.56	13	**82.50**	10	71.56	13	*	*	99
vowel	30.52	51	31.17	53	**37.01**	55	26.62	55	30.74	56	27.71	65	31.17	51	22.73	64	27.71	55	*	*	56
wdbc	85.16	15	88.89	15	**92.06**	14	88.89	10	91.53	11	90.48	9	90.48	14	83.07	12	89.95	14	89.95	12	96
wine	65.52	15	65.52	16	60.34	20	89.66	4	**94.83**	5	87.93	4	65.52	16	67.24	17	65.52	16	89.66	7	100

Table 2: Results of various distance metrics. *A* is the generalization accuracy and *H* is the number of hidden neurons generated. Reported Best is the best generalization reported in the literature. The best generalization accuracy among different distance metrics are shown in bold face.