A DISCRETE APPROACH TO CONSTRUCTIVE NEURAL NETWORK LEARNING

Justin Fletcher and Zoran Obradović

School of Electrical Engineering and Computer Science
Washington State University, Pullman WA 99164-2752 U.S.A.

ABSTRACT

Constructive algorithms have the objectives of improved generalization and simplified learning through dynamic creation of a problem-specific neural network architecture. Here, a parallel learning algorithm which constructs such an architecture is proposed. The algorithm consists of three phases: search through examples for points near the decision boundary; generation of a pool of candidate hyperplanes for boundary approximation; and selection of the separating hyperplanes from the candidate pool. The form of the final architecture is specified by the cardinality of the selected set of hyperplanes, where each individual hyperplane determines connection strengths for one hidden unit of the constructed network. While the algorithm might be too computationally demanding for a sequential implementation, the analytical expressions show that speed-up linear in the number of processors is achievable on distributed or highly parallel systems. The experimental benchmark results on a distributed network of DECStations and the Paragon supercomputer using *p4* are in agreement with analytical speed-up estimates and the architecture constructed by this algorithm is small resulting in improved generalization.

Keywords — machine learning, neural networks, parallel processing, constructive algorithms.

1 INTRODUCTION

A *neural network* is a weighted graph of simple processing units (or *neurons*). The interconnection graph of a *feed-forward* network is acyclic with processing units arranged in multiple layers consisting of input, zero or more hidden, and output layers.

All units in any layer are fully connected to the succeeding layer. Units compute an *activation function* of their weighted input sum. Here we consider *binary neural networks* where the activation function of each unit is of the form $g(x) : \mathrm{R} \rightarrow \{0, 1\}$,

$$g(x) = \begin{cases} 0 & \text{if } x < t \\ 1 & \text{if } x \geq t \end{cases}$$

where x is the unit's weighted input sum.

Traditional neural network learning (e.g. backpropagation (Rumelhart et. al., 1986)) involves modification of the interconnection weights between neurons on a pre-specified network. Determining the network architecture is a challenging problem which currently requires an expensive trial-and-error process. In selecting an appropriate neural network topology for a classification problem, there are two opposing objectives. The network must be large enough to be able to adequately define the decision boundary and should be small enough to generalize well (Geman et. al., 1992). Rather than learning on a pre-specified network topology, a *constructive algorithm* also learns the topology in a manner specific to the problem. The advantage of constructive learning is that it automatically fits network size to the data without overspecializing which often yields better generalization. Examples include the tiling algorithm of (Mézard & Nadal, 1989) and the cascade-correlation algorithm of (Fahlman & Lebiere, 1990).

This research was inspired by Baum's efficient constructive algorithm which determines separating hyperplanes between the data classes (Baum, 1991). In a single hidden layer feed-forward binary neural network, each hidden unit with fan-in k is a representation of a k-1 dimensional hyperplane. The hyperplane corresponding to the hidden unit may be determined through solution of the equation system defined by k points on the hyperplane. In Baum's algorithm a series of oracle queries is used in conjunction with training examples to determine these k points on the separating hyperplanes. The learner is allowed to ask an oracle for the correct class associated with arbitrary points in the problem domain in addition to using the training examples provided. The hyperplanes are sequentially determined by partitioning the problem domain space using training examples and queries. The hidden units of a single hidden layer feed-forward binary neural network and corresponding connections are then created from the hyperplanes. The connection weights from the hidden layer to the output layer are determined by an algorithm which separates the hidden layer representation of the problem by a single hyperplane (e.g. the perceptron algorithm (Rosenblatt, 1962)).

While Baum's algorithm is applicable where an oracle for the classification of any given point exists, in many cases such an oracle is not available or may be too expensive for practical use. In our previous work (Fletcher, 1993a) a constructive algorithm similar to Baum's but without an oracle is integrated with prior symbolic knowledge. The hybrid system gives better generalization than each component individually.

In this paper, we propose a three phase construction from examples alone. First, a search for points on the decision boundary is performed. Secondly, a pool of candidate hyperplanes for boundary approximation is obtained from the boundary points. Finally, separating hyperplanes are selected from the candidate pool. The result is a significant decrease in the complexity of the constructed network with a corresponding increase in generalization ability.

In Section 2 we describe our constructive learning algorithm. In Section 3 parallelization details are examined followed by experimental results.

2 NETWORK CONSTRUCTION

Construction of the neural network is performed in three phases:

1. Determination of points on the decision boundary;
2. Generation of a pool of candidate hyperplanes for boundary approximation from the obtained points; and
3. Selection of the final separating hyperplanes from the candidate pool and creation of hidden units from selected hyperplanes.

The remainder of this section describes each of these phases in detail.

2.1 Determination of Points on the Decision Boundary

For all pairs of training examples belonging to different classes, a search for corresponding points on the boundary separating those examples is performed. For simplicity of explanation, we will assume that the examples from the training set T belong to two classes T_1 and T_2.

Approximations to the points on the decision boundary are determined by repeatedly interpolating between example points of the classes T_1 and T_2. The interpolation begins by selecting two examples $a \in T_1$, $b \in T_2$. The *unknown region* between a and b is defined as the circle centered at the midpoint of segment defined by a and b with a diameter of the distance between a and b, as shown in Figure 1. The unknown region between a and b is then searched for the training example nearest to the midpoint of the segment defined by a and b. If such an example q is found and $q \in T_1$ (T_2) the search is then repeated in the smaller unknown region between between q and b (a). The next unknown region is shown in Figure 2.

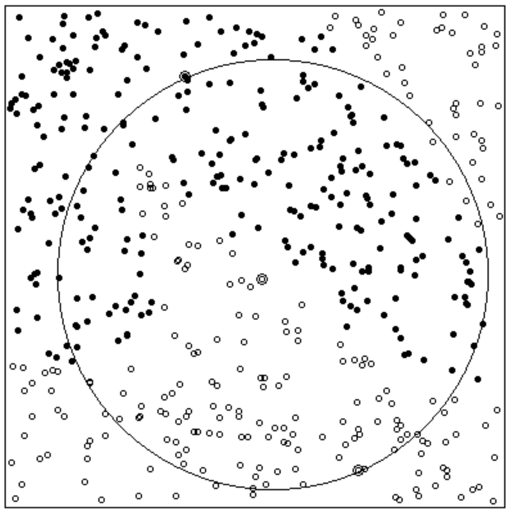
Figure 1: First unknown region

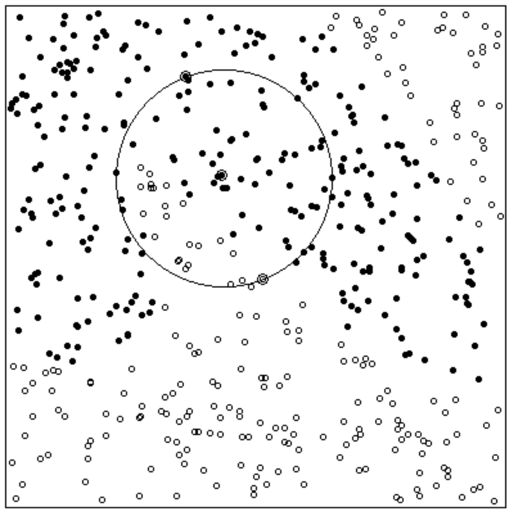
Figure 2: Next unknown region

If no point from T is found in the current unknown region, its midpoint is the closest approximation to a point on the decision boundary. If radius of this known region is within a specified tolerance the boundary point is stored providing it has not been previously determined. Boundary points continue to be generated until a predetermined number have been found or a number of data points have been examined without finding a new point on a decision boundary. The resultant boundary points are shown in Figure 3. Observe that this procedure requires two adjustable parameters. However, in practice, the algorithm is not too sensitive to these parameters providing they result in an oversized pool of boundary points.

This search for the boundary points can be parallelized by assigning different portions of the input space to available processors. While feasible, for a reasonable data distribution such a step is unnecessary as each point on the decision boundary can be found in a relatively small expected time if the training data is organized in a k-d tree structure (Bentley, 1975). In a k-dimensional space, the computational cost for organization of t training examples into a k-d tree is bounded by $O(kt \log t)$ steps. Assuming a non-malicious distribution (e.g. uniform) on the set of training examples, the expected search time for a boundary point is bounded by $O(\log t)$ unknown region reductions since an average reduction eliminates more than half of the t training examples from consideration. Using the constructed k-d tree, in each step the nearest training example to the center of an unknown region can be determined in expected time $O(\log t)$. Thus, for a non-malicious distribution, the expected time required to search for n boundary point is bounded by $O(kt \log t + \log^2 t)$. Consequently, the resources required for parallelization can be best applied to candidate hyperplane

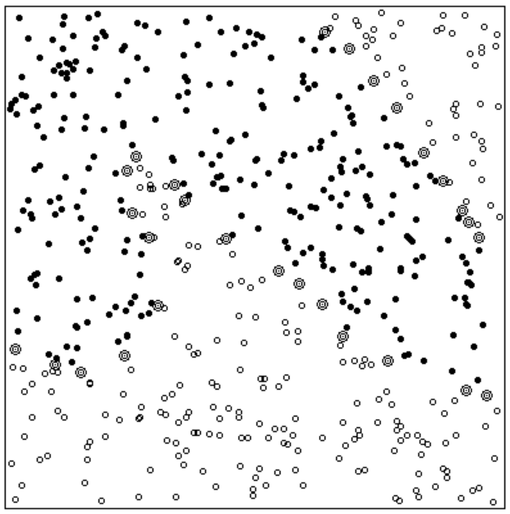
Figure 3: Boundary points

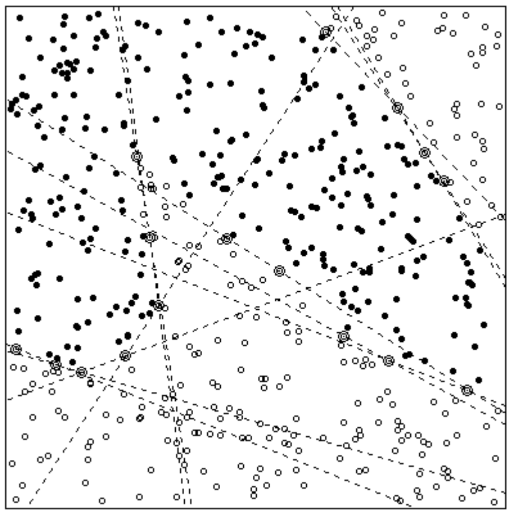
Figure 4: Candidate hyperplanes

selection as described in Section 3.

2.2 Generation of Candidate Hyperplane Pool

Once the points on the decision boundary have been found, the k-1 nearest boundary points are determined for each. As previously, k is the domain dimensionality. A pool of hyperplanes is then determined through solution of the equation system defined by each set of the k boundary points. Each unique hyperplane is saved as a candidate hyperplane (Figure 4). Observe that if n boundary points are found in phase 1, the total number of candidate hyperplanes m generated using this construction is at most n and at least n/k.

This construction of a pool of hyperplanes is motivated by the reasonable assumption that it is more probable for neighboring boundary points than for distant boundary points to define a hyperplane that approximates the optimal decision boundary. Alternatively, more hyperplanes can be constructed by using a less restrictive rule for a hyperplane determination. However, a large pool of candidates requires significantly higher computational costs in selecting appropriate candidates in the next phase of the algorithm.

2.3 Hyperplane Selection

The first hidden unit is created from the candidate hyperplane which best classifies the training data. For simplicity, we assume that two classes are of reasonably balanced size when a reasonable classification criteria is the smallest number of clas-

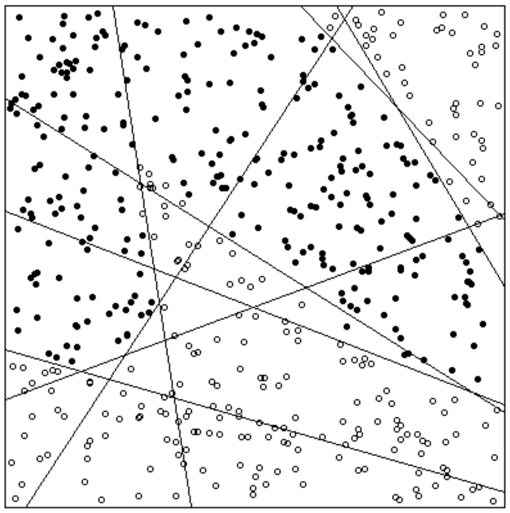
Figure 5: Selected hyperplanes

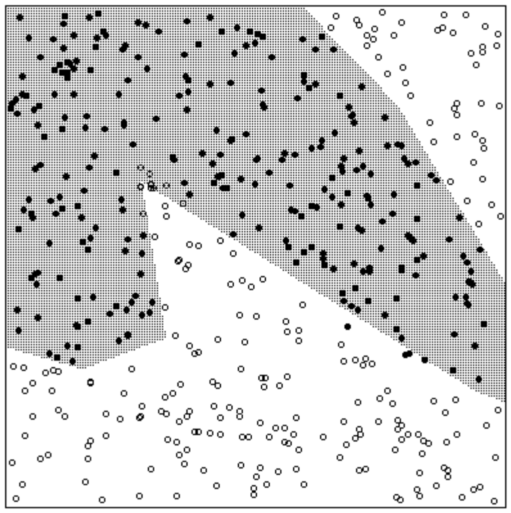
Figure 6: Decision boundary

sification errors on the training set data. The first created unit is removed from the candidate list and each remaining hidden unit is created by parallel evaluation of the remaining candidate hyperplanes in conjunction with the previously created hidden units. This is accomplished by creating a hidden unit and iteratively setting the input layer connection weights to the corresponding equation of each of the candidate hyperplanes.

The output layer weights of a candidate for the next intermediate network are determined by learning from the training examples. This task can be performed by a number of methods since it modifies only the output layer parameters with no changes to hidden units parameters. In selecting a training method for this task, one should observe that the hidden layer units are generated from examples alone and consequently may not correspond to the optimal separating hyperplanes. As such, the hidden layer problem representation of the generated network may not be linearly separable. Here a variant of the perceptron algorithm that keeps the best set of weights in a "pocket" while the perceptron is trained incrementally is used. This approach, the pocket algorithm (Gallant, 1990) yields with high probability the optimal separation for non-linearly separable problems.

Once an intermediate network has been constructed, its performance is evaluated by determining the classification accuracy on the training set. The candidate hyperplane which results in an intermediate network with the greatest performance improvement on the training set is selected and a hidden unit is created from the selected hyperplane. Given m candidate hyperplanes generated in phase 2, $O(m)$ evaluations are performed to select a hyperplane corresponding to a hidden unit in

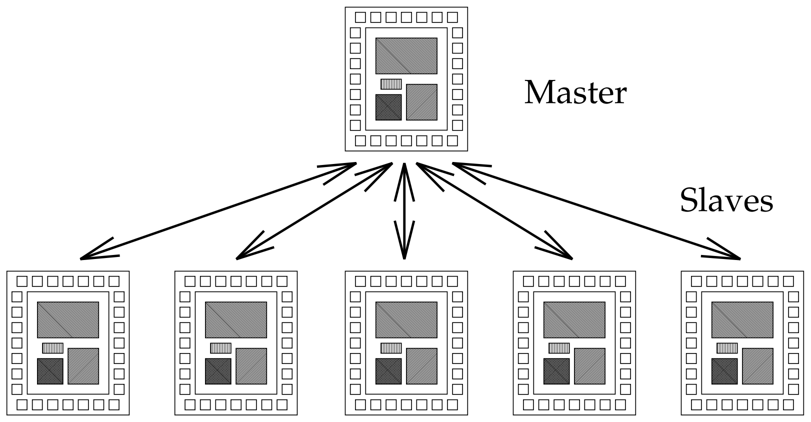

Figure 7: Parallel architecture

the network. Hidden layer construction is completed when no candidate hyperplane results in a significant improvement in classification on a cross validation set. Consequently, in selecting the final neural network architecture $O(m^2)$ evaluations are performed.

The final determination of the output layer weights are determined by use of the pocket algorithm. A final selection of hidden layer hyperplanes is shown in Figure 5 with the resultant decision boundary depicted in Figure 6.

3 EXPERIMENTAL RESULTS

In the following sections, we first propose and analyze parallelization and determine the computational speedup. We then examine the complexity of the constructed networks and test generalization on several well-known problems.

3.1 Parallel Computation

On a sequential computer, significant resources are required for the multiple evaluation of candidate hyperplanes. However, this expensive operation can be performed in parallel. Figure 7 shows the proposed parallel architecture for candidate hyperplane selection. Each processor may be either a workstation in a distributed environment or a processor on a parallel machine. One processor is responsible for the master process. This process distributes the training data at initialization. During the selection of separating hyperplanes from the candidate pool, the master process requests the slaves to create hidden units corresponding to the selected hyperplanes. While there are more candidate hyperplanes available, each slave process requests candidates to evaluate. The evaluations performed by the slave processes are integrated by the master process. If any new hyperplane is selected, a new hidden unit corresponding to the hyperplane is generated. As each slave process requests a hyperplane to eval-

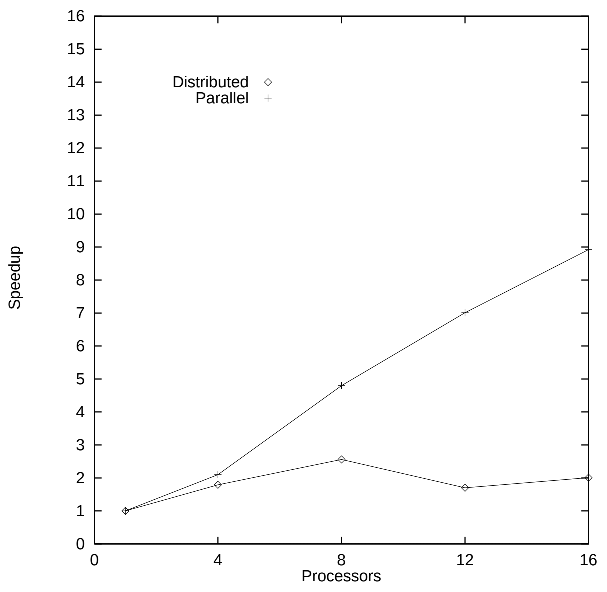

Figure 8: MONK's Problem 1 Speedup

uate rather than being assigned an even distribution of hyperplanes, automatic load balancing is performed.

In Section 2.3 we established that number of evaluations required to select a hyperplane is linear in the size of the candidate hyperplane pool. Consequently, the speedup of parallel evaluation is linear in the number of available processors assuming the size of the candidate pool is greater than the number of processors. Under this assumption parallelization provides a near-linear speedup as candidate hyperplane evaluation is the most expensive step of the algorithm.

The algorithm was implemented using *p4* (Butler & Lusk, 1992). Developed at Argonne National Laboratory, *p4* supports parallel programming for both distributed environments and highly parallel computers. Two implementation platforms were used: a local distributed system of DECStations and the Paragon at the San Diego Supercomputer Center. The Paragon (San Diego Supercomputer Center, 1993) is an Intel high-speed concurrent multicomputer consisting of 416 nodes in a mesh archi-

	DECStation	DECStations	Paragon	Paragon
Processors	1	16	1	16
Problem 1	365.65	181.02	1324.80	148.52
Problem 2	1334.52	357.72	794.02	70.25
Problem 3	3952.60	726.12	2245.42	195.22

Table 1: MONK's Problems: Total Execution Time (seconds)

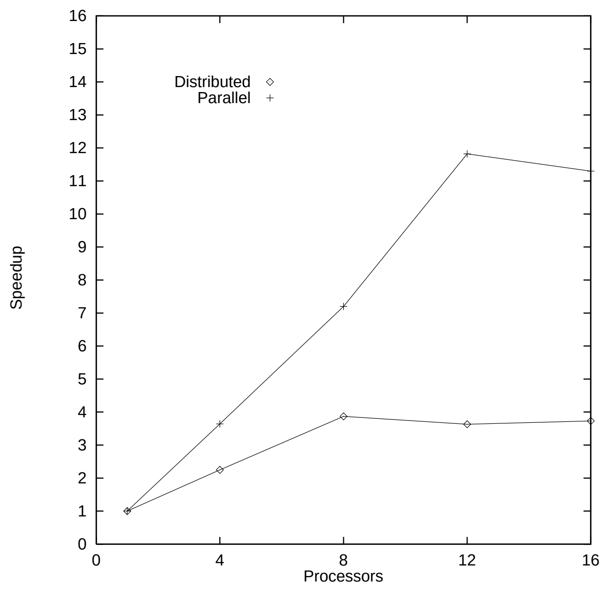

Figure 9: MONK's Problem 2 Speedup

tecture where each node contains two i860 processors, one for computation and one for communications. Of these, 16 nodes were allocated for our experiments. Development under *p4* allowed identical code to be used for the Paragon and the DECStation network.

Candidate hyperplane selection and total execution times for the MONK's problems (described in section 3.2) are shown in Table 1. Figures 8, 9 and 10 show the overall speedup for distributed and parallel environments. The overall speedup is lower in the distributed environment due to higher communication costs and the additional time required for slave process startup.

		Sequential	Distributed	Parallel
	Processors	1	16	16
Problem 1	Candidates	76.8	76.8	66.5
	Hidden Units	3.4	3.7	5.7
Problem 2	Candidates	79.0	79.0	74.2
	Hidden Units	5.9	6.4	7.1
Problem 3	Candidates	69.9	69.9	66.1
	Hidden Units	5.8	6.0	6.2

Table 2: MONK's Problems: Network Construction

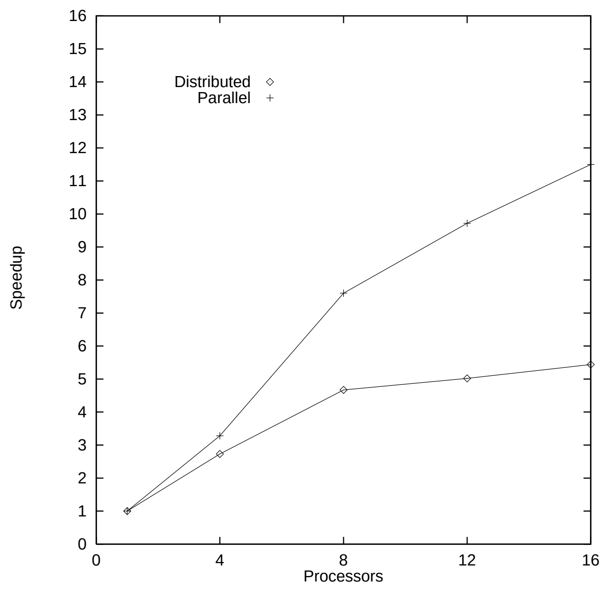

Figure 10: MONK's Problem 3 Speedup

3.2 Generalization

Experiments were performed on five well-known benchmark problems for learning systems. Three of these are highly-structured human constructed problems (the MONK's tests (Thrun et. al., 1991) and two are real-life problems (the 1984 U.S. Congressional Voting Records Database (Schlimmer, 1987; Murphy & Aha, 1994; Congressional Quarterly Inc., 1985) and the Ljubljana breast cancer domain (Zwitter & Soklic, 1988).

In MONK's problems robots are classified in one of two groups based on six multi-valued attributes. MONK 1 problem is in disjunctive normal form (DNF), MONK 2 combines attributes in a way which makes it complicated to describe in DNF or CNF using the given attributes only, and MONK 3 is in DNF in the presence of five percent of noise. In MONK problems only the classification of a subset of 124, 169 and 122 robots respectively is used for training. The learning task is to generalize using these

	Sequential		Distributed		Parallel	
Processors	1		16		16	
Accuracy	Train	Test	Train	Test	Train	Test
Problem 1	99.51	99.14	100.00	100.00	97.90	94.86
Problem 2	74.91	68.05	75.79	67.73	74.79	67.68
Problem 3	96.72	93.77	96.72	93.49	97.86	93.37

Table 3: MONK's Problems: Percentage Accuracy

	Average	Minimum	Maximum
Boundary Points	100	100	100
Candidate Hyperplanes	55.40	43	62
Hidden Units	1.50	1	5
Training Set Accuracy	94.50	92.33	96.17
Test Set Accuracy	93.55	84.09	100.00

Table 4: Voting Problem: Network Construction and Percentage Accuracy

patterns. Each problem was learned ten times with random initializations by the sequential, distributed and parallel implementations. Each of the ten experiments were limited to the determination of 100 points on the decision boundary. Table 2 shows the average number of generated candidate hyperplanes and the constructed hidden units for sequential, distributed and parallel implementations. Observe that in all cases the hyperplane selection phase drastically reduced size of the candidate pool to a small number of selected hyperplanes with their corresponding constructed hidden units. The classification and generalization abilities of the three implementations each averaged over ten runs are compared in Table 3. The results on different platforms vary slightly since the algorithm is stochastic and each system is using a different random number generator. This result is a significant gain in generalization ability over the two-phase algorithm where units are constructed directly from the generated pool of hyperplanes. In such an approach, performance on the training set is comparable to results shown in Table 3 but generalization is 85.42%, 70.37% and 72.92% for problems 1, 2 and 3 respectively (Fletcher, 1993b).

The 1984 Congressional Voting Records Database contains the records of 16 key votes for 435 Congressmen as identified by the Congressional Quarterly Almanac. Each congressman is recorded as voting yes, no or having no definitive vote recorded. Given this information, the task is to determine whether the congressman is a Republican or Democrat. The results of a ten-fold cross-validation experiment with a distributed environment of 16 DECStations is shown in Table 4. These generalization results are somewhat better and the constructed network is smaller compared to results obtained by other constructive algorithms. In (Romaniuk & Hall, 1993), cascade-correlation was reported to have 91.3% generalization while a divide and conquer neural network achieved 93.4% generalization. Both algorithms generated an average of three hidden units over the ten experiments.

The breast cancer domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Slovenia, courtesy of M. Zwitter and M. Soklic.

	Average	Minimum	Maximum
Boundary Points	90	90	90
Candidate Hyperplanes	72.12	67.60	78.80
Hidden Units	4.20	3.20	5.00
Training Set Accuracy	78.82	77.27	79.37
Test Set Accuracy	70.48	66.07	73.76

Table 5: Breast Cancer: Network Construction and Percentage Accuracy

This data set contains 286 examples each with nine features, some linear and some nominally valued. The learning task is to predict whether cancer will recur following treatment. Previous results (Michalski et. al., 1986; Clark & Niblett, 1987; Tan & Eshelman, 1988; Cestnik et. al., 1987) have a reported accuracy between 66 and 78%. Using ten five-fold cross-validation experiments, our constructive approach performs comparably as shown in Table 5.

4 CONCLUSIONS

The generalization ability of a classification and prediction system depends on its complexity. In traditional neural network systems interconnection strengths are learned from examples on a prespecified architecture. Consequently, the learning process is manually iterated by varying the architecture until one of appropriate complexity is found. In contrast, a constructive algorithm dynamically creates the appropriate network architecture required for a specific problem.

The algorithm presented here constructs such an architecture by determining a pool of candidate hyperplanes to approximate the decision boundary between classes in the example data. Once a candidate pool is determined, a small number of hidden units are created from selected hyperplanes. While the algorithm requires significant computational resources, linear speedup in the number of processors in processors is possible through parallelization. The increasing availability of networked workstation environments make this algorithm practical.

As the algorithm learns not only the connection weights but also creates the required architecture the usual manual iterative process is no longer required. The final constructed architecture is both small and problem-specific resulting in improved generalization.

ACKNOWLEDGEMENTS

This research was sponsored in part by the National Science Foundation under grant NSF-IRI-9308523 and the San Diego Supercomputing Center under grant WSU500.

REFERENCES

1. Baum, E. B., (1991), Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, v.2, pp.5–19.
2. Bentley, J. L., (1975), Multidimensional binary search tree used for associative searching. *Communications of the ACM*, v.18, pp.509–517.
3. Butler, R. & Lusk, E., (1992), *User's Guide to the p4 Parallel Programming System.*
4. Cestnik, G., Konenenko, I. & Bratko, I, (1987), Assistant-86: A knowledge-elicitation tool for sophisticated users. In Bratko, I. & Lavrac, N., editors, *Progress in Machine Learning*, pp. 31–45, Sigma Press.
5. Clark, P., & Niblett, T., (1987), Induction in noisy domains. In *Progress in Machine Learning (from Proceedings of the Second European Working Session on Learning)*, pp. 11–30, Bled, Slovenia, Sigma Press.
6. Congressional Quarterly Inc., (1985), *Congressional Quarterly Almanac.* v.XL. Washington, D.C., 98th Congress, 2nd session 1984.
7. Fahlman, S., & Lebiere, C., (1990), The cascade-correlation learning architecture. In Touretzky, D. S. editor, *Advances in Neural Information Processing Systems*, v.2, pp. 524–532, Morgan Kaufmann, San Mateo, CA.
8. Fletcher, J., & Obradović, Z., (1993), Combining prior symbolic knowledge and constructive neural network learning. *Connection Science*, v.5, pp. 365–375.
9. Fletcher, J., & Obradović, Z., (1993), Parallel and distributed systems for constructive neural network learning. In *Proc. IEEE Second Int. Symp. High Performance Distributed Computing*, pp. 174–178, IEEE Computer Society Press, Los Alamitos, CA.
10. Gallant, S. I., (1990), Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, v.1, pp. 179–191.
11. Geman, S., Bienstock, E. & Doursat, R., (1992), Neural networks and the bias / variance dilemma. *Neural Computation*, v.4, pp. 1–58.
12. Mézard, M. & Nadal, J. P., (1989), Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, v.22, pp. 2191–2204.

13. Michalski, R. S., Mozetic, I., Hong, J. & Lavrac, N., (1986), The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. Fifth National Conf. on Artificial Intelligence*, pp. 1041–1045, Morgan Kaufmann.

14. Murphy, P. M., & Aha, D. W., (1994), *UCI Repository of Machine Learning Databases.* Dpt. of Information and Computer Science, University of California, Irvine, CA. [Machine-readable data repository at ics.uci.edu].

15. Romaniuk, S. G., & Hall, L. O., (1993), Divide and conquer neural networks. *Neural Networks*, v.6, pp. 1105–1116.

16. Rosenblatt, F., (1962), *Principles of Neurodynamics.* Spartan, New York, NY.

17. Rumelhart, D., Hinton, G., & Williams, R., (1986), Learning internal representations by error propagation. In Rumelhart, D. & McClelland, J., editors, *Parallel Distributed Processing*, v.1, chapter 8, pp. 318–362. MIT Press, Cambridge, MA.

18. San Diego Supercomputer Center, (1993), *Parallel User Guide.* San Diego, CA.

19. Schlimmer, J. C., (1987), *Concept Acquisition Through Representational Adjustment.* PhD thesis, Dpt. of Information and Computer Science, University of California, Irvine, CA.

20. Tan, M., & Eshelman, L., Using weighted networks to represent classification knowledge in noisy domains. In *Proc. Fifth Int. Conf. on Machine Learning*, pp. 121–134, Ann Arbor, MI.

21. Thrun, S. B. et al., (1991), The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU–CS–91–197, Dpt. of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

22. Zwitter, M., & Soklic, M., (1988), *Breast cancer data.* Institute of Oncology, University Medical Center, Ljubljana, Slovenia. Acquired through (Murphy & Aha, 1994).

References

[1] Eric B. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2(1):5–19, January 1991.

[2] Ralph Butler and Ewing Lusk. *User's Guide to the p4 Parallel Programming System*, November 1992.

[3] G. Cestnik, I. Konenenko, and I. Bratko. Assistant-86: A knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*, pages 31–45. Sigma Press, 1987.

[4] P. Clark and T. Niblett. Induction in noisy domains. In *Progress in Machine Learning (from Proceedings of the Second European Working Session on Learning)*, pages 11–30, Bled, Slovenia, 1987. Sigma Press.

[5] Congressional Quarterly Inc. *Congressional Quarterly Almanac*, volume XL. Washington, D.C., 1985. (98th Congress, 2nd session 1984).

[6] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[7] Justin Fletcher and Zoran Obradović. Combining prior symbolic knowledge and constructive neural network learning. *Connection Science*, 5(3,4):365–375, 1993.

[8] Justin Fletcher and Zoran Obradović. Parallel and distributed systems for constructive neural network learning. In *Proceedings of the Second International Symposium on High Performance Distributed Computing*, pages 174–178, Spokane, WA, July 1993. IEEE Computer Society Press, Los Alamitos.

[9] S. Geman, E. Bienstock, and R. Doursat. Neural networks and the bias / variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[10] M. Mézard and Jean-Pierre Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22:2191–2204, 1989.

[11] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, Philadelphia, PA, 1986. Morgan Kaufmann.

[12] P. M. Murphy and D. W. Aha. *UCI Repository of Machine Learning Databases.* Department of Information and Computer Science, University of California, Irvine, CA, 1994. [Machine-readable data repository at ics.uci.edu].

[13] Steve G. Romaniuk and Lawrence O. Hall. Divide and conquer neural networks. *Neural Networks*, 6(8):1105–1116, 1993.

[14] Frank Rosenblatt. *Principles of Neurodynamics.* Spartan, New York, 1962.

[15] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge, 1986.

[16] San Diego Supercomputer Center, San Diego, CA. *Parallel User Guide*, July 1993.

[17] Jeffrey C. Schlimmer. *Concept Acquisition Through Representational Adjustment.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1987.

[18] M. Tan and L. Eshelman. Using weighted networks to represent classification knowledge in noisy domains. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 121–134, Ann Arbor, MI, 1988.

[19] Sebastian B. Thrun et al. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU–CS–91–197, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[20] Matjaz Zwitter and Milan Soklic. *Breast cancer data.* Institute of Oncology, University Medical Center, Ljubljana, Slovenia, 1988. Acquired through [12].