Constructively Learning a Near-Minimal Neural Network Architecture

Justin Fletcher and Zoran Obradović

Abstract— **Rather than iteratively manually examining a variety of pre-specified architectures, a constructive learning algorithm dynamically creates a problem-specific neural network architecture. Here we present an revised version of our parallel constructive neural network learning algorithm which constructs such an architecture. The three steps of searching for points on separating hyperplanes, determining separating hyperplanes from separating points and selecting separating hyperplanes generate a near-minimal architecture. As expected, experimental results indicate improved network generalization.**

I. Introduction

Traditional neural networks learning (e.g. back-propagation [10]) 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 separating surface and should be small enough to generalize well [7]. 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 such 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 and Nadal [8] and the cascade-correlation algorithm of Fahlman and Lebiere [3].

This research was inspired by a constructive algorithm proposed by proposed by Baum [1] which determines the separating hyperplanes between the data classes. 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. Here 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 [9]).

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. On our previous work [4] instead of depending on an oracle to determine points on separating hyperplanes, approximations to the points on the separating hyperplanes are determined by repeatedly interpolating between example points of the various classes in the training set. A parallelized construction under p4 [2] using a network of workstations and a Touchstone Delta was recently proposed [5]. This work suggests further improvement significantly affecting constructed network complexity and generalization.

In Section 2 we describe our parallel constructive learning algorithm which generates a near-minimal architecture and does not require oracle queries. In Section 3 experimental results from this algorithm are presented.

This work was supported in part by the National Science Foundation under grant IRI-9308523.

The authors are with the School of Electrical Engineering and Computer Science, Washington State University, Pullman WA 99164-2752, USA.

Z. Obradović is also affiliated with the Mathematical Institute, Belgrade, Yugoslavia.

0-7803-1901-X/94 $4.00 ©1994 IEEE

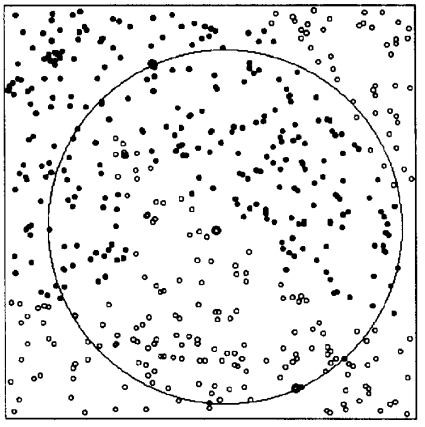

Figure 1: First unknown region

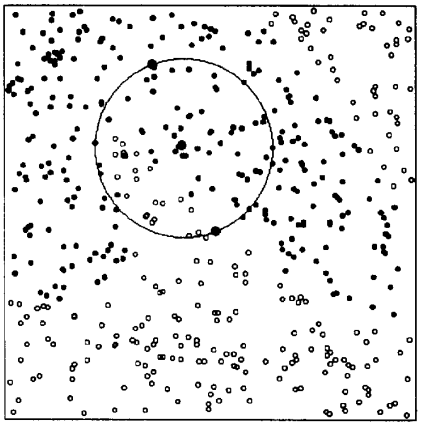

Figure 2: Next unknown region

II. Network Construction

In our previous work [5], separating hyperplanes were determined sequentially and thus previously determined points on the separating hyperplanes were not maintained. As a result, effort was wasted by repeatedly determining the same points on the hyperplanes. This modification of the algorithm maintains all points on the separating hyperplanes. This was accomplished by modifying the algorithm to initially determine the separating points and then create the candidate hyperplanes. The best candidates are then selected through an iterative process resulting in a near-minimal architecture.

Construction of the neural network is thus performed in three stages:

1. Determination of points on separating hyperplanes
2. Determination of candidate hyperplanes from points on the separating hyperplanes and
3. Creation of hidden units from selected hyperplanes.

A. Determination of Separating Points

For all pairs of training examples belonging to different classes, a search for corresponding points on the hyperplanes separating those examples is done. This search for the separating points can be performed in parallel. Each separating point is found as follows. Approximations to the points on the hyperplane are initially determined by repeatedly interpolating between example points of the various classes T_1 and T_2 in the training set T. The interpolation begins by selecting positive and negative examples $m \in T_1$, $n \in T_2$. The unknown region between m and n is then searched for the nearest point $q \in T$ to the midpoint of m and n. The unknown region is defined as the the circle centered at the midpoint of m and n with a diameter of the distance between m and n, as shown in Figure 1. If q is found, the search is then repeated in the smaller unknown region between q and m or q and n respectively depending on whether q is positive or negative (Figure 2).

If no point from T is found in the current unknown region, its midpoint p^1 is the closest approximation to a point on the separating hyperplane. If the distance from p^1 to an endpoint is within a specified tolerance the separating point is stored. The resultant separating points are shown in Figure 3.

B. Determination of Candidate Hyperplanes

Once the separating points hyperplanes have been determined, the k-1 nearest separating points are found for each separating point where k is the input dimensionality. Hyperplanes are determined through solution of the equation system defined by the k points on the hyperplane. Each unique hyperplane is saved as a candidate hyperplane (Figure 4). Observe that if n separating points are determined in step 1, the total number of candidate hyperplanes using this construction is at most n.

The construction is motivated by the hope that nearest separating points define the same hyperplane. Alternatively, one can construct more hyperplanes using some less restrictive rule for a hyperplanes determination. However, a large pool of candidates means significantly higher computational costs in selecting appropriate candidates in the next step of the algorithm.

C. Hyperplane Selection

The first hidden unit is created from the candidate hyperplane which best classifies the training data.

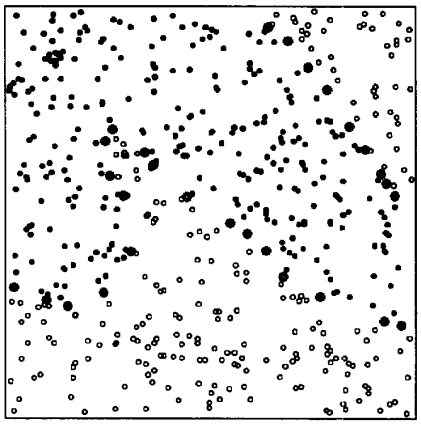

Figure 3: Determined separating points

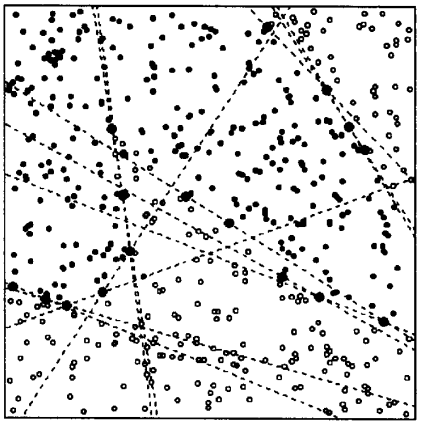

Figure 4: Candidate hyperplanes

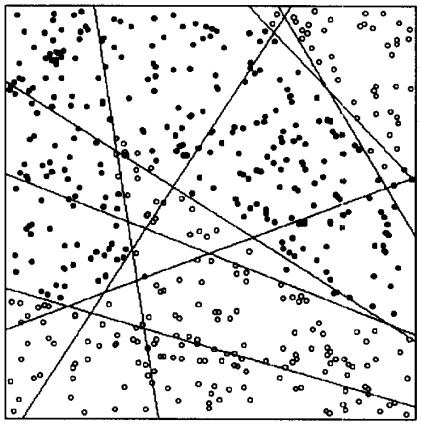

Figure 5: Selected hyperplanes

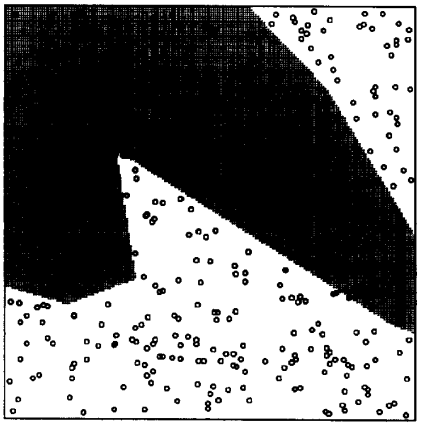

Figure 6: Separating surface

This hyperplane is then removed from the candidate list. The remaining hidden units are created by parallel evaluation of each 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 the candidate hyperplanes. The output layer weights for a candidate for the next intermediate network are then determined from the training examples. The candidate hyperplane which results in an intermediate network with the greatest performance improvement on the training set is added to the selected hyperplanes. Given n candidate hyperplanes, $O(n)$ evaluations are performed to select a hyperplane corresponding to a hidden unit in the network. Hidden layer construction is completed when no candidate hyperplane results in a significant improvement in classification on the training set. Consequently, $O(n^2)$) evaluations are performed in selecting the final neural network architecture. A final selection of hidden layer hyperplanes is shown in Figure 5 with the resultant separating surface depicted in Figure 6. The master process then performs the final task of training the output layer weights as explained in the following section.

D. Output Layer Weights

The hidden layer units are generated from examples alone, and so they may not correspond to the optimal separating hyperplanes. As such, the hidden layer problem representation of the generated network may not be linearly separable. The pocket algorithm [6] is a single-layer neural network learning algorithm that finds the optimal separation un-

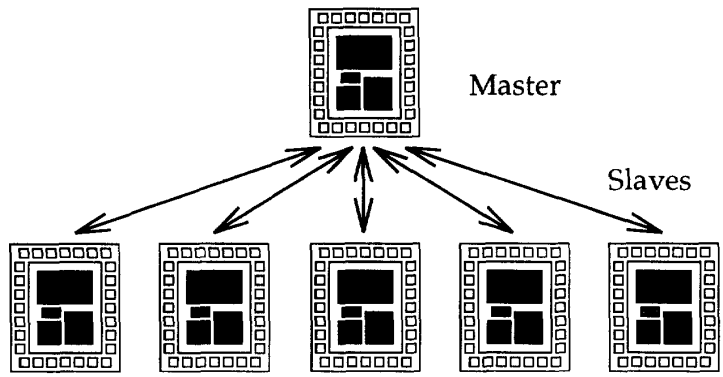

Figure 7: Parallel architecture

Accuracy	Average Number of Hidden Units	Average		Best Case		Worst Case	
		Train	Test	Train	Test	Train	Test
Problem 1	4.8	99.19	96.66	100.00	100.0	97.58	88.43
Problem 2	6.3	73.60	66.73	78.11	68.98	62.13	63.89
Problem 3	5.7	96.14	94.00	98.36	97.22	93.44	87.50

Table 1: Generated Architecture and Percentage Accuracy on the MONK's Problems

Processors	Sequential		Distributed		Parallel	
	1		19		64	
Accuracy	Train	Test	Train	Test	Train	Test
Problem 1	100.00	85.42	100.00	81.02	100.00	81.71
Problem 2	81.07	70.37	85.80	72.45	91.72	75.23
Problem 3	96.72	72.92	99.18	77.08	100.00	78.94

Table 2: Previous Percentage Accuracy on the MONK's Problems

der a given topology for problems that are not linearly separable. This algorithm keeps the best set of weights in the "pocket" while the perceptron is trained incrementally. This variant of the perceptron algorithm is used to determine the output layer weights of the intermediate networks.

E. Implementation

As significant computational resources are required due to the multiple evaluation of candidate hyperplanes this operation is performed in parallel. Figure 7 shows the 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 master process distributes the training data at initialization. During the selection of separating hyperplanes from the candidate pool, the master process informs the slaves to create hidden units corresponding to the selected hyperplanes. While there are more candidate hyperplanes available, each slave processor requests candidates to evaluate. The evaluations performed by the slave processors are integrated by the master processor. If any new hyperplane is selected, a new hidden unit corresponding to the hyperplane is generated. A system with a balanced computational load is obtained by this distribution of the candidate hyperplane analysis.

III. Experimental Results

Experiments were performed on the MONK's problems [11] using a sequential implementation to compare the quality of generalization between the previous and current constructive algorithm. The MONK's problems consist of three six-feature binary classification problems which represent specific challenges for standard machine-learning algorithms, such as the ability to learn data in disjunctive normal form, parity problems and performance in the

presence of noise.

Table 1 show results from ten runs of a sequential simulation of this algorithm. Significant performance gains over previous algorithm (Table 2) are obtained for problems 1 and 3. Note that as predicted by neural network theory, network generalization is best with the fewest number of generated units as excess hidden units may result in data overfitting.

IV. Conclusions

Neural networks efficiency and prediction quality depend significantly on how we select the network architecture, learning algorithm and initial set of weights. The constructive learning algorithm presented here learns not just the connection weights but also creates the required near-minimal architecture resulting in improved generalization. Parallel implementation using p4 is under development. Further study will be performed on large scale benchmarks and real-life problems such as protein secondary structure prediction.

References

[1] E. 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] R. Butler and E. Lusk. User's guide to the p4 parallel programming system. Technical Report ANL–92/17, Argonne National Laboratory, Argonne, IL, 1992.

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

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

[5] J. Fletcher and Z. 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.

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

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

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

[9] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

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

[11] S. 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.