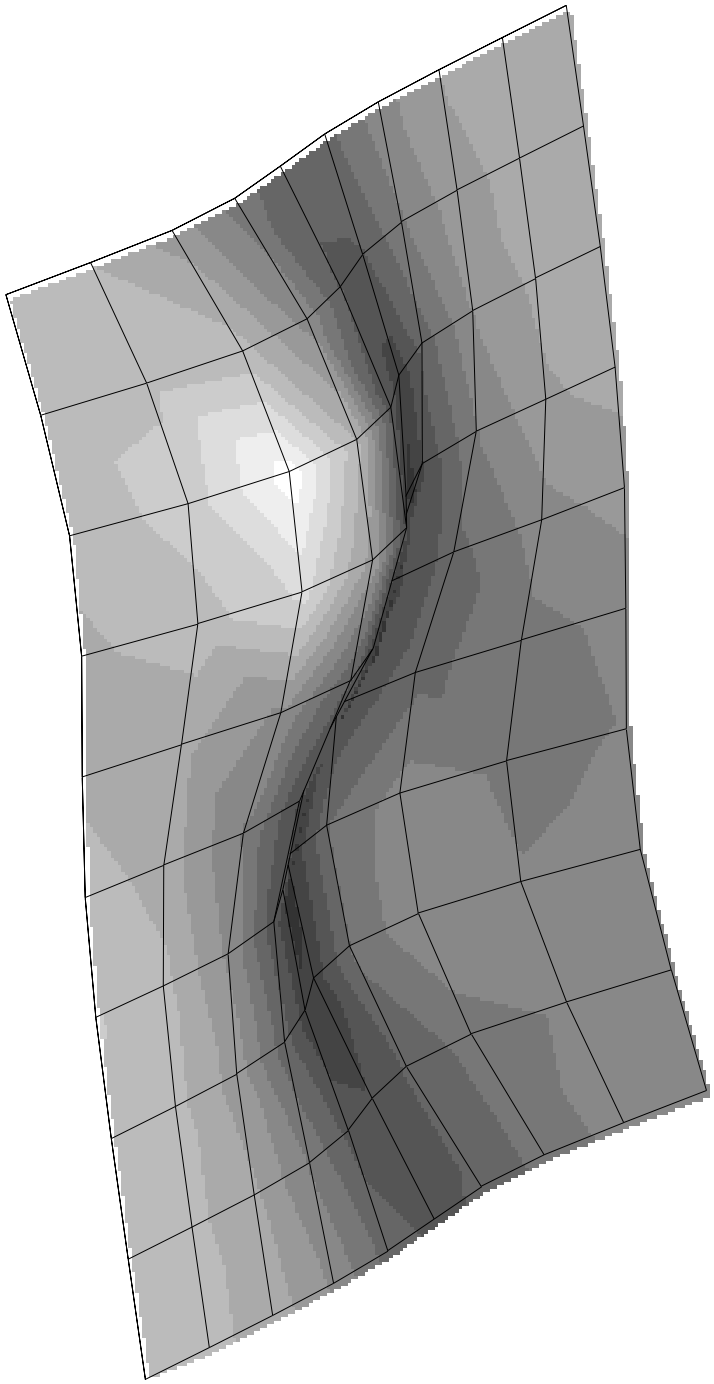

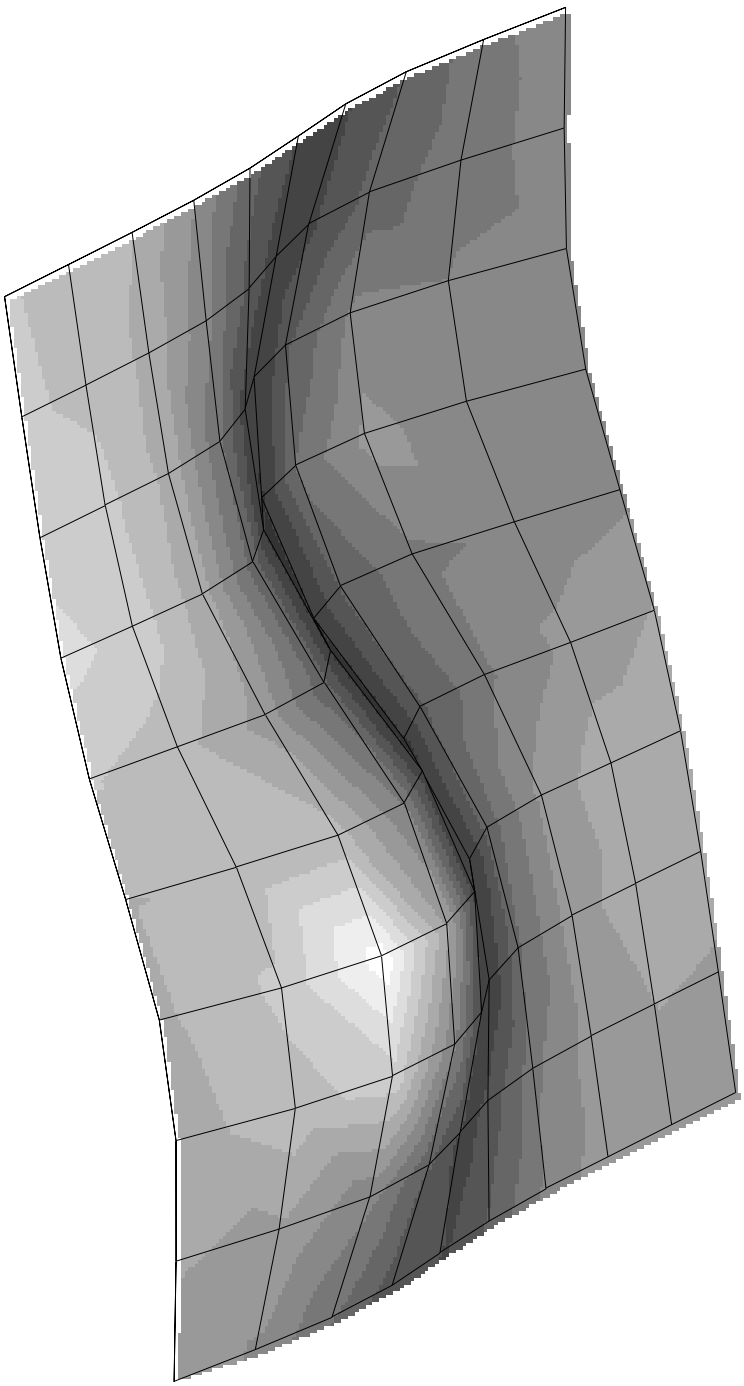

and 3 *OFF* weights of 0.03. The eighth node had *ON* weights of 1.39 to 1.41, and its *OFF* weights were all between 0.02 and 0.05. After an additional 1000 learning cycles this trial, too, had learned essentially perfect response.

Rather similar results were obtained when a nine nodes rather than eight were used in the system, although on the average the learning was somewhat slower. Eight of the nodes learned to respond to a distinct vertical or horizontal line, and the remaining *extra* node learned approximately equal weights from all inputs. In six of ten trials this pattern of response was learned within 5000 learning cycles; all trials learned it within 15,000 cycles of learning.

Progressively increasing values of the weight-limiting parameter were used for smaller numbers of nodes. With fewer than eight nodes, the system could only partially separate the lines. In all the the trials described below, results are given after 5000 cycles. Seven nodes learned to produce a few nodes which respond to exactly one input line. Up to six of the seven nodes, but on the average three nodes, learned such a response. An average of 1.5 additional nodes learned a rougher approximation to this; they learned to respond strongly to one line as well as weekly to another line of perpendicular orientation. The other 2.5 nodes, on the average, learned to respond rather strongly both to one horizontal and one vertical line.

As the number of nodes is reduced further this last type of response becomes more typical. With only four nodes, the strong response to both one vertical and one horizontal line became the universal pattern for each node. In addition, the response of one node to its two lines became more nearly equal in strength.

Reducing the number of nodes to three brought a different behavior. Two of the nodes learned to respond to one line, either horizontal or vertical, but always in the same direction, and the third line learned to respond strongly to the remaining two lines in that direction. With only two nodes the results were less clean, but one node learned to respond strongly to two of the lines, either horizontal or vertical, and the other node learned to respond strongly to the other two lines in the same orientation.

Thus, given sufficient nodes, the Competitive Hebbian algorithm produces perfect separations. The system learned a set of appropriate strong but distinct responses entirely from the structure present in the input examples.

5 Conclusion

The learning algorithm introduced here, Competitive Hebbian Learning, has been demonstrated to show valuable properties in that multiple nodes trained by this algorithm develop complementary responses to a training set. The algorithm is an important extension of Hebbian learning which seems clearly superior to previous competitive algorithms.

Acknowledgement

The author would like to thank David Zipser and the Department of Cognitive Science at UCSD for their hospitality during his sabbatical leave in Spring, 1990. Without Dr. Zipser's suggestions, advice, and encouragement this paper would have been impossible.

The figures show results of Competitive Hebbian Learning with two nodes after 30,000 training cycles, when the weights were very stable for these trials. Two nodes is the most interesting case. For this trial the sum of weights-squared was limited to 289. The two nodes appear to have learned to divide the input space along the pair of diagonals of the input square, essentially creating a pair of orthogonal axes. The two nodes are quite precisely orthogonal. The square of node-zero output, summed over the input array, is 3.199. For node one the similar sum of output-squared is 3.191. In contrast, the sum over the array of node-zero output times node-one output, the dot product, is only 0.00304, one-tenth percent of the squared response of either node.

Tests with three nodes used a sum of weights-squared limit of 144, and for four nodes it was 16. In each case, the nodes developed strong responses in different regions of the input space, essentially sharing that space quite equally. The nodes in each case are roughly "orthogonal," but not to the precision possible with two nodes. For three nodes the dot products of nodes are $1-2\%$ of the squared node response, and for four nodes the dot products are $10-20\%$ of squared node response.

These results show that Competitive Hebbian Learning does produce valuable multiple-node responses for this simple test. The results are best for two nodes, reflecting the two-dimensional structure of the training set. Further nodes cannot find additional structure in the input, and are thus forced to learn to divide the input space.

4 Separating Perpendicular Lines

In this section the problem is to test the system on separating "horizontal lines" and "vertical lines" in an input space which is treated as a square array of inputs. For this test there are 16 inputs, and in each training presentation exactly 4 of these inputs are *ON*. For example, the first four inputs *ON* means the top row is *ON*.

The initial weights are chosen to have small random positive values and all the input values are non-negative, $+1/2$ for *ON* or 0 for *OFF*, so the system is monopolar. Since the last term in the modified Hebb rule is limited to non-negative values, all weights remain non-negative, and all outputs are also non-negative. For these trials the weight-limit function is chosen to be linear, so the limit is of the form $\sum_{j \in I} \mid w_j \mid = W_{sum} = Constant$.

Since there are four possible horizontal lines and four possible vertical lines, the Competitive Hebbian Learning system obtains the cleanest results with a system of eight output nodes, one node for each of the possible lines. Using a weight-limit parameter W_{sum} of 6.0, ten independent trials were run for 5000 learning cycles. In each trial one node learned to respond to each of the horizontal lines and one node learned to respond to each of the vertical lines.

A node has learned perfectly when four of its weights are exactly 1.50 for the four inputs which make up a certain line, and the remaining weights are exactly zero. In nine of the ten trials the error with respect to ideal weights was less than 0.005 after 5000 learning cycles. In the other trial six of the eight nodes had learned essentially perfectly, as in the other nine trials. A seventh node had 3 *ON* weights of 1.47 rather than 1.50,

change the sign of Δw_{ij} we impose a limiting value of zero when $[1-4\sum_{l\neq i} y_l^2]$ is negative. This means that no learning occurs in such cases.

Competitive Hebbian Learning involves a lateral, inter-node inhibition on the changes in the w_{ij}, the *feedforward* interconnections in the system. The systems considered in this paper are strictly feedforward systems and these sytems thus have no dynamic properties. There is no learning of lateral weights required. A hardware implementation, or, for that matter, a biological implementation, could utilize fixed lateral connections to implement the lateral inhibition on the learning in a strictly local manner.

In addition, the Competitive Hebbian Learning algorithm may be extended by adding trainable internode lateral weights. Such a system would have learnable dynamic properties, not limited by the Competitive Hebbian Learning on the feedforward weights.

To summarize, the algorithm has these significant properties:

1. A set of N nodes may learn, simultaneously, N distinct components of the training-set input.

2. A strictly local implementation is possible.

3. The input-output response of the system may be strictly feedforward, as in the applications considered in this paper.

4. Dynamic systems, with trainable dynamics, may be produced by adding trainable lateral internode weights.

3 Random Spots

For a first test simulation of the Competitive Hebbian algorithm, take a system of a few nodes which respond to a set of 100 inputs, imagined as a regular 10 by 10 square array. Each input presentation was Gaussian spot, centered at a random position at least two inputs away from the nearest edge in the input array. For each input presentation the spot had the same strength, and the background level was adjusted so that the average input value was zero.

The idea of this test is that, while a single Hebbian node should learn to respond to the overall structure of the training set, multiple Competitive Hebbian nodes should learn to share the input space effectively. For this test, as for most of the simulations presented in this paper, a limit on the sum of squares of the weights into each node was used. Weights were initialized to small random values. The training set consisted of 5000 different randomly positioned spots, and the learning-rate parameter, α, was set at 0.15.

The first trial used one node, simple Hebbian learning, with a limit of 625 on the sum of squares of weights into that node. The expected symmetric response, strongest in the center, developed within 5000 cycles and was essentially unchanging by 10,000 training cycles. The learned response was symmetric and strongest in the center region.

Next, take the partial derivatives of G with respect to the w_{ij}:

$$\frac{\partial G}{\partial w_{ij}} = f_i^{'}[2y_i x_j + 2y_i x_j \sum_{k \neq i} \lambda_{ik} (y_k)^2]. \tag{5}$$

To optimize G make the Δw_{ij} proportional to the corresponding partial derivative of G, dividing by $f_i^{'}$ first:

$$\Delta w_{ij} = \alpha y_i x_j [1 + \sum_{k \neq i} \lambda_{ik} y_k^2]. \tag{6}$$

This is simply Hebbian learning multiplied by the term in square brackets.

Still to be determined are the Lagrange multipliers λ_{ik}. Consider the change in any one of the g_{ik} produced by changes in the w_{ij}:

$$\Delta g_{ik} = \sum_j [\frac{\partial g_{ik}}{\partial w_{ij}} \Delta w_{ij} + \frac{\partial g_{ik}}{\partial w_{kj}} \Delta w_{kj}]. \tag{7}$$

This simplifies to

$$\Delta g_{ik} = 2\alpha g_{ik} (\sum_j x_j^2)[f_i^{'}(1 + \sum_{l \neq i} \lambda_{lk} y_l^2) + f_k^{'}(1 + \sum_{l \neq k} \lambda_{il} y_l^2)]. \tag{8}$$

Note that the two halves of the term in square brackets are identical except for the indices. Since $f_m^{'} = \frac{1}{4} - y_m^2$, this term has a particularly simple and elegant form, symmetric to second order in y_i, if all the λ_{ik} are equal to -4. With this choice of the λ_{ik} the effect of changes in the weights on each Δg_{ik} is uniform, to lowest order in node output, and independent of i and k. Making this substitution Δw_{ij} becomes

$$\Delta w_{ij} = \alpha [y_i x_j][1 - 4 \sum_{l \neq i} y_l^2], \tag{9}$$

the Competitive Hebbian Learning rule.

One extremely important consideration remains. As in other associative-learning systems, weights will tend to grow larger as the system learns. A limit must be imposed on the overall strength of the input to each node. In some trials of Competitive Hebbian Learning we have used a constant limit on the sum of the squares of the weights into each node and in other trials a constant limit on the sum of the absolute values of the weights into each node. It must also be emphasized that not merely the existence of a weight limit, but indeed the actual value of this parameter, is of crucial importance in Competitive Hebbian Learning. The value of this parameter, in relation to the other parameters of the system, determines how effectively the system will be able to learn to respond to the training-set inputs.

Note the characteristics of this modification to Hebb's rule. The modification is a multiplicative factor which reduces the learning rate when nodes other than the node into which w_{ij} connects are strongly active. In order to insure that this factor does not

allows a strictly local implementation. Thus, it may be considered as a simplified model of biological systems, applicable, for example, to models of visual processing. Furthermore, since it is local, the calculation time for serial simulations is proportional only to the number of weights. Finally, learning proceeds, in principle, in parallel on all the nodes.

2 The Competitive Hebbian Algorithm

Consider first Hebbian learning for a set of nodes with common inputs. A set of N nodes y_i are non-linear functions of a set of I inputs x_j. For each node write

$$y_i = f_i(\sum_{j=1}^{I} w_{ij} x_j) - 1/2, \tag{1}$$

where f_i is the squashing function for node i and w_{ij} is the interconnection strength, or weight, from the input j to the node i. Each squashing function f_i is the usual sigmoid, $f(u) = 1/(1 + e^{-u})$, and the constant $1/2$ is subtracted from the y_i so that the y_i run symmetrically from $-1/2$ to $+1/2$.

Define an overall response function F to be the sum of the squares of the node activities, $F = \sum_{i=1}^{N} y_i^2$. One may approach an extremum of F as a function of each w_{ij} by making a change in w_{ij}, Δw_{ij}, which is proportional to the partial derivative of F with respect to w_{ij}. Multiplying by a learning-rate parameter α gives

$$\Delta w_{ij} = \alpha y_i x_j, \tag{2}$$

the Hebb rule.

Unfortunately, there is nothing in Hebbian learning to cause the various y_i to learn to respond to different components of the input; that is the crux of the problem. With pure Hebbian learning on all the nodes we expect multiple nodes to learn to respond largely to a principal component of the input. What is required is a modification which forces the set of N nodes to learn to respond to distinct components of that input.

For this purpose consider a set of constraints

$$g_{ik} = (y_i y_k)^2 = 0, \tag{3}$$

where i and k are any members of the set of nodes such that $i \neq k$, and S is some constant. The constraints $g_{ik} = 0$ require that nodes i and k be orthogonal. Note that the sign of the node activity does not enter.

Now form the function G which is obtained by multiplying each distinct g_{ik} by a Lagrange multiplier λ_{ik} and then adding the sum to F. Setting λ_{ik} for $k < i$ equal to λ_{ki}, this can be written in the symmetric form

$$G = \sum_{i=1}^{N} y_i^2 + \frac{1}{2} \sum_{i=1}^{N} \sum_{k \neq i} \lambda_{ik} g_{ik}. \tag{4}$$

To be presented at IJCNN-91 Seattle

Competitive Hebbian Learning

Ray H. White
Departments of Physics and Computer Science
University of San Diego
San Diego, California 92110

January 21, 1991

Abstract

Of crucial importance for applications of unsupervised learning to systems of many nodes with a common set of inputs is how the nodes may be trained to collectively develop optimal response to the input. In this paper Competitive Hebbian Learning, a modified Hebbian-learning rule, is introduced. In Competitive Hebbian Learning the change in each connection weight is made proportional to the product of node and input activities multiplied by a factor which decreases with increasing activity on the other nodes. The individual nodes learn to respond to different components of the input activity while collectively developing maximal response. Several applications of Competitive Hebbian Learning are then presented to show examples of the power and versatility of this learning algorithm.

1 Introduction

It is widely recognized that Hebbian learning is a learning process which trains a single node in a feedforward system to develop an optimal response to a set of inputs. The problem when one wishes to train a *set* of nodes to optimally respond to a common set of inputs is that one wants the individual nodes to respond to distinct components of the input activity, that is the response of each node should effectively complement the responses of the other nodes. In this paper a new modified Hebbian learning algorithm is introduced and applied to several different problems. This new algorithm, Competitive Hebbian Learning, has the valuable property that different nodes learn simultaneously to respond to different features of the training set, even as each node develops a strong response to that training set.

Among the important characteristics of Competitive Hebbian Learning is the fact that strictly feedforward multi-node systems are possible; the examples presented in the later sections of this paper are such feedforward systems. The training interaction among the nodes introduces no unwanted dynamics, nor does it limit the dynamics if lateral inter-node connections are added for problems which include dynamics. The algorithm also