Learning, invariance, and generalization in high-order neural networks

C. Lee Giles and Tom Maxwell

High-order neural networks have been shown to have impressive computational, storage, and learning capabilities. This performance is because the order or structure of a high-order neural network can be tailored to the order or structure of a problem. Thus, a neural network designed for a particular class of problems becomes specialized but also very efficient in solving those problems. Furthermore, *a priori* knowledge, such as geometric invariances, can be encoded in high-order networks. Because this knowledge does not have to be learned, these networks are very efficient in solving problems that utilize this knowledge.

I. Introduction

A key facility of neural networks is their ability to recognize invariances, or to extract essential parameters from complex high-dimensional data. The process of recognizing invariants requires the construction of complex mappings, which utilize internal representations, from a higher dimensional input space to a lower dimensional output space. The internal representations of these mappings capture high-order correlations which embody the invariant relationships between the input data stream and the output.

Typically, an internal representation is constructed from a group of first-order units via a learning rule such as backpropagation. First-order units are units which are linear in the sense that they can capture only first-order correlations. They can be represented by

$$y_i(x) = S\left[\sum_j^N W(i,j)x(j)\right],$$

where $\mathbf{x} = \{\mathbf{x}(j)\}$ is an input vector (bold type indicates vectors), $W(i,j)$ is a set of adaptive parameters (weights), N is the number of elements in the input vector $\mathbf{x}$, and S is an unspecified sigmoid function.

We suggest that the use of building blocks which are able to capture high-order correlations is a very efficient method for building high-order representations. In many cases, greatly enhanced performance in learning, generalization, and knowledge (symbol) representation can be achieved by crafting networks to reflect the high-order correlational structure of the environment in which they are designed to operate. Our research in this area has focused on high-order units which can be represented by the equations

$$y_i(x) = S\left[W_0(i) + \sum_j W_1(i,j)x(j) + \sum_j \sum_k W_2(i,j,k)x(j)x(k) + \ldots\right],$$

where the higher-order weights capture higher-order correlations. A unit which includes terms up to and including degree k will be called a kth order unit.

We have investigated various topics with high-order networks: learning and generalization, implementation of invariances, associative memory, temporal sequences, and computational feasibility of high-order approaches.[1–5] This paper is an introduction to the forenamed work and that work should be consulted for additional details and explanations.

II. Background

Early in the history of neural network research it was known that nonlinearly separable subsets of pattern space can be dichotomized by nonlinear discriminate functions.[6] Attempts to adaptively generate useful discriminate functions led to the study of threshold logic units (TLUs). The most famous TLU is the perceptron,[7] which in its original form was constructed from randomly generated functions of arbitrarily high order. Minsky and Papert[8] studied TLUs of all orders, and came to the conclusions that high-order TLUs were impractical due to the combinatorial explosion of high-order terms, and that first-order TLUs were too limited to be of much interest. Minsky and Papert also showed that single feed-forward slabs of first-order TLUs can implement only linearly separable mappings. Since most problems of interest are not linearly separable, this is a very serious limitation. One alternative is to cascade slabs of first-order TLUs. The units embedded in the cascade (hidden units) can then combine the outputs of previous units and generate nonlinear maps. However, training in cascades is very difficult[7] because there is no simple way to provide the hidden units with a training signal. Multislab learning rules require thousands of iterations to con-

C. L. Giles is with U.S. Air Force Office of Scientific Research, Bolling Air Force Base, Washington, DC 20332, and T. Maxwell is with Sachs/Freeman Associates, 1401 McCormick Drive, Landover, Maryland 20785.

Received 14 August 1987.

verge, and sometimes do not converge at all, due to the local minimum problem.

These problems can be overcome by using single slabs of high-order TLUs. The high-order terms are equivalent to previously specified hidden units, so that a single high-order slab can now take the place of many slabs of first-order units. Since there are no hidden units to be trained, the extremely fast and reliable single-slab learning rules can be used.

More recent research involving high-order correlations includes optical implementations,[9,10] high-order conjunctive connections,[11] sigma-pi units,[12] associative memories,[4,5,9,18] and a high-order extension of the Boltzmann machine.[13]

III. High-Order Neural Networks

A high-order neuron can be defined as a high-order threshold logic unit (HOTLU) which includes terms contributed by various high-order weights. Usually, but not necessarily, the output of a HOTLU is (1,0) or (+1,−1). A high-order neural network slab is defined as a collection of high-order logic units (HOTLUs). A simple HOTLU slab can be described by

$$y_i = S[\text{net}(i)]$$
$$= S[T_0(i) + T_1(i) + T_2(i) + T_3(i) + \ldots + T_k(i)], \quad (1)$$

where y_i is the output of the ith high-order neuron unit, and S is a sigmoid function. $T_n(i)$ is the nth order term for the ith unit, and k is the order of the unit. The zeroth-order term is an adjustable threshold, denoted by $W_0(i)$. The nth order term is a linear weighted sum over nth order products of inputs, examples of which are

$$T_1(i) = \sum_j W_1(i,j)x(j), \qquad T_2(i) = \sum_j \sum_k W_2(i,j,k)x(j)x(k), \quad (2)$$

where $x(j)$ is the jth input to the ith high-order neuron, and $W_n(i,j,\ldots)$ is an adjustable weight which captures the nth order correlation between an nth order product of inputs and the output of the unit. Several slabs can be cascaded to produce multislab networks by feeding the output of one slab to another slab as input. (The sigma-pi[12] neural networks are multilevel networks which can have high-order terms at each level. As such, most of the neural networks described here can be considered as a special case of the sigma-pi units. A learning algorithm for these networks is generalized back-propagation. However, the sigma-pi units as originally formulated did not have invariant weight terms, though it is quite simple to incorporate such invariances in these units.)

The learning process involves implementing a specified mapping in a neural network by means of an iterative adaptation of the weights based on a particular learning rule and the network's response to a training set. The mapping to be learned is represented by a set of examples consisting of a possible input vector paired with a desired output. The training set is a subset of the set of all possible examples of the mapping. The implementation of the learning process involves sequentially presenting to the network examples of the mapping, taken from the training set, as input–output pairs. Following each presentation, the weights of the network are adjusted so that they capture the correlational structure of the mapping. A typical single-slab learning rule is the perceptron rule, which for the second-order update rule can be expressed as

$$W'_2(i,j,k) = W_2(i,j,k) + [t(i) - y(i)]x(j)x(k). \quad (3)$$

Here $t(i)$ is the target output and $y(i)$ is the actual output of the ith unit for input vector $\mathbf{x}$. Similar learning rules exist for the other W_i terms. If the network yields the correct output for each example input in the training set, we say that the network has converged, or learned the training set. If, after learning the training set, the network gives the correct output on a set of examples in the training set that it has not yet seen, we say that the network has generalized properly.

Another learning rule which we have investigated is the outer product rule, sometimes referred to as the Hebbian learning rule. In this form of learning, the correlation matrix is formed in one shot via the equations:

$$W_2(i,j,k) = \sum_{s=1}^{Np} [y^s(i) - \mathbf{y}(i)][x^s(j) - \mathbf{x}(j)][x^s(k) - \mathbf{x}(k)],$$

$$\mathbf{y}(i) = \left[\sum_{s=1}^{Np} y^s(i)\right]/Np, \qquad \mathbf{x}(j) = \left[\sum_{s=1}^{Np} x^s(j)\right]/Np,$$

where Np denotes the number of patterns in the training set. The training set is denoted by $\{(x^s,y^s)|s \in (1,Np)\}$, and $\mathbf{y}$ and $\mathbf{x}$ are the averages of y^s and x^s over the training set. In cases in which $\mathbf{x}$ is nearly zero [which often is the case when (+1,−1) input encoding is used], simplicity may be gained in the above equations at the expense of a small reduction in performance by setting the $\mathbf{x}$ and $\mathbf{y}$ terms to zero. This illustrates the reduction in computation often achieved with the proper choice of data representation. (Unless otherwise specified, all training patterns discussed in this paper are represented by +1's in a −1 background.) Analogous equations hold for other orders.

IV. Learning the EXCLUSIVE-OR

Learning the EXCLUSIVE-OR (see below) has been the classic difficult problem for first-order units because it is the simplest nonlinearity separable problem. Training a hidden unit to perform this function requires thousands of iterations of the fastest learning rules.

EXCLUSIVE-OR

$x(1)$	$x(2)$	$t(x)$
1	1	1
1	−1	−1
−1	1	−1
−1	−1	1

An alternative method is to use a fixed hidden unit. An equivalent approach is to train a second-order unit

to solve this problem. The training set consists of the EXCLUSIVE-OR values given above. Consider a second-order unit of the form:

$$y(\mathbf{x}) = \mathrm{sgn}[W_1(1)x(1) + W_1(2)x(2) + W_2x(1)x(2)], \quad (4)$$

with the learning rules:

$$W_1'(i) = W_1(i) + [t(x) - y(x)]x(i),$$

$$W_2' = W_2 + [t(x) - y(x)]x(1)x(2).$$

Here, $t(x)$ is the correct output for input $x = (x_1,x_2)$. The signum function becomes sgn[x], which is defined as $+1$ when $x > 0$ and -1 when $x < 0$. The second-order term is equivalent to a handcrafted hidden unit, so that the single second-order unit takes the place of a two-layer cascade. Since there is no correlation between x_1 or x_2 and the desired output $t(x)$, the $W_1(i)$ terms average to zero in the learning process. However, since $x(1)x(2)$ is perfectly correlated with $t(x)$, W_2 is incremented positively at each step of the learning procedure, so that the training process converges in one iteration to the solution $W_1(i) = 0$ and $W_2 > 0$. One iteration is defined as training of the network over all members of the training set. This high-order unit will learn an arbitrary binary map in one iteration of the above learning rule.

V. Implementation of Invariances

In high-order networks it is possible to handcraft the units such that their output is invariant under the action of an arbitrary finite group of transformations on the input space.[2] A unit is invariant under the action of a transformation group $G = \{g\}$*iff*:

$$y(gx) = y(x) \text{ for all } g \in G. \quad (5)$$

This invariance is imposed by averaging the weight matrices over the group, thus eliminating the unit's ability to detect correlations which are incompatible with the imposed group invariance. In particular, we can handcraft a G-invariant unit from the HOTLU described by Eq. (1). This handcrafted unit is described by the equation

$$y(x) = S\left[\sum_{g\in G} \mathrm{net}(gx)\right], \quad (6)$$

where the sum is over all members of the group G and the net operator is defined as in Eq. (1). To see that this unit is invariant under the group G, note that for $h \in G$, we have

$$y(hx) = S\left[\sum_{g\in G} \mathrm{net}(ghx)\right] = S\left[\sum_{g'\in G} \mathrm{net}(g'x)\right] = y(x), \quad (7)$$

where we have made the substitution $g' = gh$. The second step follows from the fact that a sum over $g = g'h^{-1}$ is equivalent to a sum over g', since multiplying all terms in a sum over a group by a member of the same group simply results in a permutation of the terms in the sum. In some important cases, this averaging process allows a collapse of the weight matrix when redundant terms are eliminated, analogous to the reduction in dimension which occurs in the transformation from absolute to relative coordinates.

As an example of the power of using the group invariance operators, we will use the sign-change group as an alternative solution to the EXCLUSIVE-OR problem. The EXCLUSIVE-OR is invariant under the action of the sign-change group, members of which are denoted by $\{S_i, i \in \{1,2\}\}$:

$$S_0\{x(j)\} = x(j), \qquad S_1\{x(j)\} = -x(j).$$

Averaging the sign-change group over the second-order unit of Eq. (4) yields the simplification

$$y(x) = S\left\{\sum_{j=1}^{2} W_1(j)[x(j) - x(j)] + W_2[x(1)x(2) + x(1)x(2)]\right\}$$

$$= S[2W_2\,x(1)x(2)].$$

The problem is essentially solved; all that remains is to determine the sign of W_2.

Let us now illustrate the implementation of translation invariance in a more intuitive fashion. Suppose we are given a second-order unit described by the equation

$$y(x) = S\left[\sum_j \sum_k W_2(j,k)x(j)x(k) + \ldots\right].$$

We wish to determine what constraints must be placed on the W matrix in order to ensure that the unit's output y is invariant under the transformation group T. Define $T = \{g(m)\}$ such that $g(m)x(k) = x(m + k)$. If we apply this group operator to the previous equation, the result is

$$y[x] = y[g(m)x] \text{ for all } g(m) \in T.$$

Expanding the right-hand side of this equation yields

$$y[g(m)x] = S\left[\sum_j \sum_k W_2(i,j,k)x(j + m)x(k + m) + \ldots\right]$$

$$= S\left[\sum_j \sum_k W_2(i,j - m,k - m)x(j)x(k) + \ldots\right],$$

where in the last step we have assumed either that periodic boundary conditions are appropriate or that edge effects are negligible. A comparison of the above equations reveals that the constraint that must be placed on W_2 to ensure translation invariance is

$$W_2(j,k) = W_2(j - m,k - m) \text{ (translation invariance constraint).}$$

To understand this condition, let us note that W_2 is a function of a pattern composed of two points located at j and k. This constraint stipulates that W_2 have the same value when evaluated at any pattern $(j - m,k - m)$ which is a translated version of the original pattern (j,k). In other words, to ensure translation invariance, W_2 must depend only on the equivalence class of patterns under translation; then the weight function $W_2(j,k)$ will not distinguish between any shifted pattern in the input space. Such patterns are described as translation-equivalent. This constraint can be generalized to implement any invariance under an arbitrary transformation group.

Another way of expressing the invariance constraint derived in the previous paragraph becomes clear if we note that the set of translation equivalence classes of

two point patterns is given by the relative coordinate $dj = j - k$. In other words, every possible value of dj corresponds to a single equivalence class and vice versa. We now note that this condition is equivalent to constraining W_2 such that it depends only on the relative coordinate dj and not on the absolute coordinates j and k (which would distinguish between different patterns belonging to the same equivalence class). This leads to the condition on W_2 that

$$W_2(j,k) \rightarrow W_2(dj) \text{ (translation invariance solution).}$$

Thus, the implementation of translation invariance in Eq. (1) is equivalent to redefining the correlation matrices such that they depend only on relative position dj and not absolute position j. The first- and second-order invariant terms are

$$T_1(i) = W_1(i) \sum_j x(j), \quad T_2(i) = \sum_{dj} W_2(i,dj) \sum_j x(j)x(j + dj). \quad (8)$$

The expression for T_1 is not very interesting. A sum over the input is, of course, translation-invariant. The expression for T_2 should not be surprising. Optical processing has long known and used the translation-invariant properties of the autocorrelation. What is different here is that arbitrary scale factors W_1 and W_2 are being learned by the neural network and W_2 weights the autocorrelation. Optical implementations of this approach are discussed in Ref. 10.

An error-correcting perceptron learning rule for the second-order term can be written as

$$W_2'(i,dj) = W_2(i,dj) + [t(i) - y(i)] \sum_j x(j)x(j + dj). \quad (9)$$

An invariant outer product or Hebbian rule can also be easily formulated. A single unit may have both invariant and noninvariant terms. Many other invariances (including rotation and scale) can be implemented in this manner. Simulations indicate that prior implementation of known invariances can be a very powerful step in the adaptation of a network to a problem environment.

We have implemented translation-invariant networks for both pattern recognition[2,3] and associative memory[4,5] applications. The associative network's attractors are localized patterns of activity which are independent of their absolute position on the neural slab. We conjecture that the implementation of invariances may be an important step in the construction of neural network symbol processors.

VI. Competitive Learning

Competitive learning algorithms, discussed in Ref. 14, have the advantage that they are capable of classifying input patterns without requiring a training signal $t(i)$ as in the supervised learning rules discussed above. The drawback of most of these algorithms is that there is no way to tell *a priori* if the classification categories generated by the network will be useful or interesting. In this section we discuss a method, based on group theory, for handcrafting classification categories for high-order competitive learning networks.

Since a group G-invariant HOTLU yields a response that is dependent only on the equivalence class of the input pattern, such a unit will naturally function as a pattern classifier, with its classification categories determined by the group G. Geometrical feature detectors can be constructed in this way, provided we can define the features as invariants of transformation groups. Constructing an architecture in which each unit of the network responds to a distinct feature involves creating competition between the feature detectors, together with training rules which prevent one unit from capturing many features. Rules of this sort are discussed in Ref. 14. Here we discuss an application in which different units naturally capture distinct categories.

Consider the problem of constructing a HOTLU that will discriminate between horizontal and vertical lines. Since these two pattern classes belong to different equivalence classes under the translation group, a translation-invariant HOTLU will yield one output for all horizontal lines and another output for all vertical lines. The probability that the outputs will be the same is effectively zero given enough precision in the training. However, ambiguous patterns such as diagonal lines will cause either of the outputs to respond. The classifier network consists of two competing second-order units with linear threshold functions, described by the equations

$$y_i(x) = \sum_{dj} W_2(i,dj) \sum_j x(j)x(j + dj), \quad (10)$$

where the subscript $j = \{j_x,j_y\}$ represents position in a 2-D pattern. The tensor W_2 is initialized randomly and normalized, so that

$$\sum_{dj} W_2(i,dj) = 1 \text{ for } i \in (1,2). \quad (11)$$

Both units are fed the same binary input pattern x, where $x \in \{+1,-1\}$, displayed on an $N \times N$ grid. The input pattern set consists of a set of horizontal and vertical lines presented in random order. The output of the system, z, is arbitrarily defined to be 1 if $y_1 > y_2$ and 2 if $y_2 > y_1$. The case $y_1 = y_2$, which never occurred in our simulations, would result in an error condition and renormalization of the weights.

Simulations performed over randomly generated presentations of the training set indicate that this system (with few iterations over the training set) always yields a 1 for any horizontal lines and a 2 for any vertical lines (or vice versa). The initial weights were randomly chosen and the training set consisted of six horizontal and vertical lines presented over a 6×6 grid. The two units can now be sensitized to their respective pattern classes by updating the weights at each input pattern presentation via the learning rule. For each input pattern presentation, the unit whose output $y(i)$ is a maximum updates its weights with the learning rule:

$$W_2'[z(x),dj] = W_2[z(x),dj] + A_L \sum_j x(j)x(j + dj), \quad (12)$$

where A_L (set arbitrarily at 0.5) controls the rate of

learning. The unit whose output $y(i)$ is a minimum does not change its weights. The weights are also renormalized at each step, so that Eq. (1) remains valid for W'_2. The problem of one unit capturing both categories, which did not arise in this example, can be dealt with in general by allowing the losing HOTLU to learn also with a reduced A_L value.

The learning process endows the system with the capacity to generalize and sort arbitrary patterns according to their degree of horizontalness or verticalness. The translation invariance gives the net the capability to discriminate only between different pattern classes independent of the pattern position. The capacity of generalization, involving the clustering of equivalence pattern classes arises in this case from the learning process. We speculate that this generalization capacity could also have been implemented, however, by the imposition of a higher level invariance constraint.[11] Although in this example we considered only two-category classification, the architectures discussed can be readily expanded to classify N categories with N output units.

It is useful to compare the capabilities of this high-order translation-invariant neural net with the layered first-order network in Ref. 12. The high-order competitive network learned the problem with orders of magnitude reduction in iterations and did not require any additional special teaching patterns. Furthermore, Rumelhart, Hinton and Williams required a factor of 2 increase in the number of input and hidden units.

VII. *TC* Problem

The *TC* problem involves distinguishing between a shifted and rotated T and C. If the T and C are represented as patterns embedded in a 3×3 square array (as shown below), it is not difficult to see that one must simultaneously inspect 3 squares in the array to discriminate between these letters:

```
XXX   XXX
 X     X
 X     XXX.
```

Thus, the *TC* problem as defined here is a third-order problem.[8] The training patterns consist of single letters inscribed in 3×3 squares, each of which occurs in an arbitrary position and orientation in a larger (10×10) spatial array. These training patterns are randomly presented to the HOTLU. The correct classification can be obtained with training over only a few iterations of the training set using a pair of translation- and rotation-invariant competitive HOTLUs as discussed above. Here we will examine another approach which only learns the rotated letters by supervised perceptron-like learning. The shifted patterns are not learned in individual shifted positions, but from one arbitrary position due to the encoded translation-invariant network. In this case the solution requires a single third-order translation-invariant HOTLU with short-range interconnects (a 3×3 window). Dynamics are governed by the equation

$$yi(x) = \mathrm{sgn}\left[\sum_{dj_1}\sum_{dj_2} W_3(i,dj_1,dj_2) \sum_j x(j)x(j+dj_1)x(j+dj_2)\right], \quad (13a)$$

with the weight update rule

$$W'_3(i,dj_1,dj_2) = W_3(i,dj_1,dj_2) + [t(i) - y(i)] + \sum_j x^s(j)x^s(j+dj_1)x^s(j+dj_2). \quad (13b)$$

The dj_1 and dj_2 sums are limited to the 3×3 window. In general, for an $m \times m$ window, this architecture will require less than m^{2k-2} adaptive weights, where k is the order of the HOTLU. For initial random weights values, the training procedure converges to a correct set of weights in one to ten iterations. Note that all known methods of solving this problem involving training hidden units in cascades require more units and adaptive weights, and utilize learning rules which usually take at least thousands of iterations to converge. (An alternate approach to the TC problem is to put translation-invariant 2cd order weights in the input layer of a single hidden layer feed-forward network. The output layer weights are 1st order. Generalized back propagation solves this problem with an order of magnitude reduction in learning time compared to the learning time taken for the same network with only 1st order weights.[19])

VIII. Generalization in Neural Networks

An inductive inference problem, as characterized by Angluin and Smith,[15] consists of five components:

(1) a rule to be inferred, generally characterized by a set of examples;

(2) a hypothesis space, that is, a space of descriptions such that any admissible rule can be described by a point in the space;

(3) a training set, or a sequence of examples that constitute admissible presentations of the rule;

(4) an inference method; and

(5) the criterion for a successful inference.

Roughly speaking, the generalization problem can be stated as given a sequence of examples of the rule, use the inference method to search through the hypothesis space until a point is reached which satisfied the criterion for successful induction.

The neural network generalization problem (as defined here) is a special case of the problem outlined above. An admissible rule in this case is a mapping from a set of possible network input patterns to a set of possible network output patterns. The hypothesis space of a neural network is defined by the architecture of the network; a point in hypothesis space is characterized by a specific set of weights (or a point in weight space). The training set consists of some subset of the set of all input patterns for which the rule is defined (together with the associated outputs). The inference method is a learning rule which modifies the weights of the network, generally in response to feedback information on how well the network performed on a given

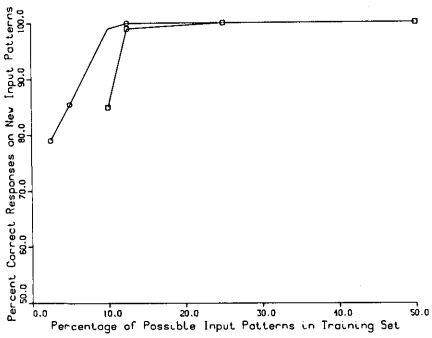

Fig. 1. Generalization capability vs percentage size of training set for the two–three clump discrimination problem using a second-order neural network. The string length is $Nx = 9$. The upper curve (circles) is for a second-order net with a second-order translation-invariant term. The lower curve (squares) is for a second-order neural network, no invariance.

example of the rule. The test of generalization consists in training the network until it performs perfectly on the training set, and then measuring its performance on a set of examples that were not included in the training set.

IX. Generalization and the Contiguity Problem

We have examined several types of contiguity problems; here we discuss one which we call the two–three clump problem. Further details and simulations are presented in Ref. 1. The performance of backpropagation on this problem has been analyzed extensively by other researchers.[16] In the two–three clump problem, the training set consists only of patterns which have either two or three clumps, contiguous strings of ones separated by non-ones (however for simulations, +1's and −1's were used). The target output of the unit is defined to be +1 for two-clump patterns and −1 for three-clump patterns:

two-clumps	three-clumps
011001100	110011011

All versions of the contiguity problem are at least second-order problems (see Ref. 8). Thus, learning these problems with a neural net will require either a cascade of first-order units or at least a second-order unit. A first-order perceptron is capable of learning the two–three clump problem with slightly more than 50% accuracy.[16] Backpropagation seems to generalize very poorly on this problem, and requires thousands of presentations in order to learn the training set.[16]

We studied the two–three clump problem using a second-order unit with terms T_0, T_1, and T_2, and the training procedure described earlier [Eqs. (1) and (2)]. After initializing the weights with random values, the unit was trained on a fraction of the data set with the range of the second-order interactions $R2 = 4$ (i.e., the range of the sum of dj over W_2 never exceeded 4) and the input pattern length $Nx = 9$. For $Nx = 9$, the number of all possible inputs is 512. After training, we made use of an iterative weight thresholding procedure to reduce the noise in the final weight values. Following the thresholding, the generalization capacity of the unit was tested by counting the number of errors that the unit made on the set of examples that were not included in the training set. In Fig. 1 we have plotted generalization capacity vs training set size for this problem. The upper curve is the result with translation invariance imposed on the second-order weight W_2, and the lower curve is the result without translation invariance imposed. Without invariance, the unit generalizes nearly perfectly for training sets as small as $\frac{1}{8}$ of all possible inputs, but drops sharply to 85% accuracy for $\frac{1}{10}$ of all possible inputs. Average convergence times ranged from three presentations of the training set for $\frac{1}{2}$ of all possible inputs to ten presentations for $\frac{1}{10}$ of all possible inputs. With invariance, the unit generalizes nearly perfectly for training sets as small as $\frac{1}{10}$ of all possible patterns, but drops sharply to 85% accuracy for training sets with $\frac{1}{20}$ of the possible patterns. The average convergence time was about three presentations for all cases with invariance. Because the net must effectively simulate a parallel edge counter to solve this problem, it appears that translation invariance does not play an important role when the training set is large.

An additional feature of this solution was that the high-order neural network unit explicitly (not implicitly as in layered networks[12]) generated an algorithmic solution to the contiguity problem. By closely inspecting the magnitudes of the weights, an explicit formula emerges. If one pays attention to the large magnitude weights and ignores the small magnitude weights, the resulting network actually generates the equation for a parallel edge-counter detector. This should not be surprising since this net has been handcrafted to solve this problem. See Ref. 1 for further details.

X. Generating High-Order Representations

The applicability of the high-order approach depends on the system designer's ability to choose a set of high-order terms which adequately represents the problem task domain which is likely to be encountered by the network. Building a network with all possible 2^N combinations of N inputs is clearly unfeasible. We have investigated several methods of choosing a representative set of terms:

(1) Matching the order of the network to the order of the problem. A single layer of kth order units will solve a kth order problem (as defined by Minsky and Papert[8]). Thus, if the order of the problem is known, we can specify the order of the unit required to solve the problem. Although the order of many hard problems such as speech recognition and visual pattern recognition is unknown, it can be estimated.

(2) Implementation of invariances. If it is known *a priori* that the problem to be implemented possesses a given set of invariances, those invariances can be preprogrammed into the network by the methods discussed earlier, thus eliminating all terms which are incompatible with the invariances. For example, speech recognition is invariant under temporal scale and translation operations.

(3) Correlation calculations. One way to determine

which terms will be useful is to calculate the correlation matrices [Eq. (4)] for a representative sampling of the mapping to be generated. The entries in the correlation matrix which are largest in absolute value correspond to terms which are highly correlated with the output of the map, and thus are most likely to make an important contribution to the network when the map is implemented. Other smaller terms can be ignored.

(4) Generating representations adaptively. It is possible to generate high-order representations by progressively adapting the network to its problem environment. There are many adaptation methods which[12] may be applicable to this problem; generalized backpropagation is an example of such a method which generates high-order representations in a different (multilayer) context. Applying gradient descent methods to the problem of generating high-order representations for single-layer architectures is very similar to the correlation calculation methods discussed under (3) above. A suggested approach is to use the capabilities of genetic search algorithms[17] for generating high-order representations.

XI. Conclusions

By choosing a set of high-order terms which embodies prior knowledge about the problem domain, we are able to construct a network which can learn and generalize very efficiently within its designated environment. By providing the network with the tools it needs to solve the problems it expects to encounter, we liberate the network from the difficult task of deciding which tools it will need and then creating those tools. This process constitutes a major part of the learning procedure for networks utilizing backpropagation. For example, in the contiguity problem, the second-order terms of the form $x(j)x(j+1)$ explicitly encode edge detection information; thus, the network only needs to learn how to use these edge terms, not generate the edge detection terms itself. Since a human uses edges to solve the contiguity problem (at least for large strings), the network is solving the problem in a similar fashion.

The drawback of the high-order approach is the combinatorial explosion of high-order terms inherent in a single-slab HOTLU. Various methods exist for dealing with this problem: cascades of high-order slabs, limitation of order, and reduced interconnections. In this paper we have discussed another method, which involves using prior knowledge of the problem domain to weed out the terms which have a small likelihood of being useful. This method produces specialized networks which are very powerful in a limited domain. We suggest that a general-purpose learning network might be constructed from a number of specialized modules designed for specific types of tasks, combined with more general networks to handle new and unusual situations.

References

1. T. Maxwell, C. L. Giles, and Y. C. Lee, "Generalization in Neural Networks: the Contiguity Problem," in *Proceedings, IEEE International Conference on Neural Networks*, San Diego (June 1987); U. Maryland Physics Department Technical Report UMLPF 87-050 (June 1987).
2. T. Maxwell, C. L. Giles, Y. C. Lee, and H. H. Chen, "Transformation Invariance Using High Order Correlations in Neural Net Architectures," IEEE Trans. Syst. Man Cybern. **SMC-86CH2364-8,** 627 (1986).
3. T. Maxwell, C. L. Giles, Y. C. Lee, and H. H. Chen, "Nonlinear Dynamics of Artificial Neural Systems," AIP Conf. Proc. **151,** 299 (1986).
4. H. H. Chen, Y. C. Lee, T. Maxwell, G. Z. Sun, H. Y. Lee, and C. L. Giles, "High Order Correlation Model for Associative Memory," AIP Conf. Proc. **151,** 86 (1986).
5. Y. C. Lee, G. Doolen, H. H. Chen, G. Z. Sun, T. Maxwell, H. Y. Lee, and C. L. Giles, "Machine Learning Using a Higher Order Correlation Network," Physica D **22,** 276 (1986).
6. N. J. Nilsson, *Learning Machines* (McGraw-Hill, New York, 1965).
7. F. Rosenblatt, *Principles of Neurodynamics* (Spartan, New York, 1962).
8. M. L. Minsky and S. Papert, *Perceptrons* (MIT Press, Cambridge, MA, 1969).
9. D. Psaltis, J. Hong, and S. Venkatesh, "Shift Invariance in Optical Associative Memories," Proc. Soc. Photo-Opt. Instrum. Eng. **625,** 189 (1986); D. Psaltis and C. H. Park, "Nonlinear Discriminant Functions and Associative Memories," AIP Conf. Proc. **151,** 370 (1986). R. A. Athale, H. H. Szu, C. B. Friedlander, "Optical Implementation of Associative Memory with Controlled Nonlinearity in the Correlation Domain," Optics Letters, vol. 11 #7, p 482 (1986). Y. Owechko, G. J. Dunning, E. Maron, and B. H. Soffer, "Holographic Associative Memory with Nonlinearities in the Correlation Domain," Applied Optics, vol. 26 #10, p 1900 (1987).
10. R. D. Griffin, C. L. Giles, J. N. Lee, T. Maxwell, and F. P. Pursel, "Optical Higher Order Neural Networks for Invariant Pattern Recognition," presented at Optical Society of America Annual Meeting (Oct. 1987).
11. G. E. Hinton, "A Parallel Computation that Assigns Canonical Object-based Frames of Reference," Proceedings of 7th International Joint Conference on Artificial Intelligence, ed. A. Drina, p 683 (1981). J. A. Feldman, "Dynamic Connections in Neural Networks," Biol. Cybern. **46,** 27 (1982). Representative papers are: D. H. Ballard, "Cortical Connections and Parallel Processing: Structure and Function," Behav. Brain Sci. **9,** 67 (1986).
12. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," Ch 8 in *Parallel Distributed Processing*, vol 1., D. E. Rumelhart and J. L. McClelland, (MIT Press, Cambridge, MA, 1986).
13. T. J. Sejnowski, "Higher-Order Boltzmann Machines," AIP Conf. Proc. **151,** 398 (1986).
14. S. Grossberg, "Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature Detectors," Biol. Cybern. **23,** 121 (1976); D. E. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning," Cognitive Sci. **9,** 75 (1985).
15. D. Angluin and C. H. Smith, "Inductive Inference: Theory and Methods," ACM Comput. Surv. **15,** 237 (1983).
16. J. S. Denker and S. A. Solla, "Generalization and the Delta Rule," presented at the Neural Networks for Computing Conference, Snowbird, UT (Apr. 1987).
17. J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard, *Induction* (MIT Press, Cambridge, MA, 1986).
18. P. Peretto and J. J. Niez, "Long Term Memory Storage Capacity of Multiconnected Neural Networks," Biological Cybernetics *54,* p 53 (1986).
19. C. L. Giles, R. D. Griffin, T. P. Maxwell, "Encoding Invariances in Higher Order Neural Networks," presented at IEEE conf. on Neural Information Processing Systems—Natural and Synthetic," Denver, Co. Nov. 1987.