Neural Networks, Vol. 1, pp. 131–139, 1988
Printed in the USA. All rights reserved.

ORIGINAL CONTRIBUTION

Applications of Counterpropagation Networks

ROBERT HECHT-NIELSEN

Hecht-Nielsen Neurocomputer Corporation

(*Received and accepted* 17 *December* 1987)

Abstract—*By combining Kohonen learning and Grossberg learning a new type of mapping neural network is obtained. This* counterpropagation *network functions as a statistically optimal self-adapting look-up table. The paper describes applications of this network to pattern recognition, function approximation, statistical analysis, and data compression.*

1. INTRODUCTION

Of all of the practical information processing operations that neural networks can currently carry out, one of the most useful is the ability to learn a mathematical mapping by adaptation in response to examples of the mapping's action. Typically, one generates a set of examples $(\mathbf{x}_1, \mathbf{y}_1)$, $(\mathbf{x}_2, \mathbf{y}_2)$, . . . of the action of a function ϕ: $\mathbf{R}^n \rightarrow \mathbf{R}^m$, where $\mathbf{y}_i = \phi(\mathbf{x}_i)$. The x_i's are assumed to be chosen in accordance with a fixed probability density function ρ. These examples serve to statistically define the desired input/output relationship.

A neural network that implements an approximation to a function or mapping is called a *mapping neural network.* Currently, the most popular mapping neural network is the backpropagation network of Rumelhart & McClelland (1986). The backpropagation network is best suited for implementing mappings that can be approximated by multiple layers of sigmoided linear combiners (which is a very rich set of functions).

In this paper some applications of a new mapping neural network (counterpropagation) are discussed. The counterpropagation network has capabilities and limitations that are considerably different from those of backpropagation. The applications discussed are some for which counterpropagation is better suited than backpropagation. This helps clarify the differences between counterpropagation and other mapping networks.

2. THE COUNTERPROPAGATION NETWORK

The counterpropagation network (CPN) architecture is a combination of a portion of the self-organizing map of Kohonen (1988) and the outstar structure of Grossberg (1969, 1971, 1982). Figure 1 presents the topology of the CPN. The basic idea is that, during adaptation, pairs of example vectors $(\mathbf{x}, \mathbf{y})$ are presented to the network at layers 1 and 5, respectively. These vectors then *propagate* through the network in a *counterflow* manner to yield output vectors $\mathbf{x}'$ and $\mathbf{y}'$ that are intended to be approximations of $\mathbf{x}$ and $\mathbf{y}$. Thus the name *counterpropagation.* For further details and discussions of related issues, see Hecht-Nielsen (1987a, 1987b, 1988a, 1988b, 1988c).

For the purposes of this paper the *forward-only* version of the counterpropagation network (shown in Figure 2) shall be used. A brief description of this network's architecture, operation and capabilities is given below.

This architecture consists of three slabs: an input layer containing n fanout units that simply multiplex the input signals $x_1, x_2, \ldots, x_n$, a middle Kohonen layer with N processing elements that have output signals $z_1, z_2, \ldots, z_N$, and a final Grossberg outstar layer with m processing element output signals $y'_1, y'_2, \ldots, y'_m$. The outputs of the Grossberg layer have primes on them because they represent approximations to the components $y_1, y_2, \ldots, y_m$ of $\mathbf{y} = \phi(\mathbf{x})$. During training, these "correct" values are supplied to the units of the Grossberg layer from the $\mathbf{y}$ input units of layer 1 (see Figure 2).

The operation of the network is usually thought of as consisting of two successive stages: *training* and *normal operation* (although these can occur simultaneously). During training, the network is exposed to examples of the mapping ϕ. It is assumed that the $\mathbf{x}$ vectors are drawn from $\mathbf{R}^n$ in accordance with a fixed probability density function ρ [**note:** the counterpropagation network used in this paper varies slightly from those discussed in Hecht-Nielsen (1987a, 1987b) in that

Requests for reprints should be sent to Robert Hecht-Nielsen, Hecht-Nielsen Neurocomputer Corporation, 5501 Oberlin Drive, San Diego, CA 92121.

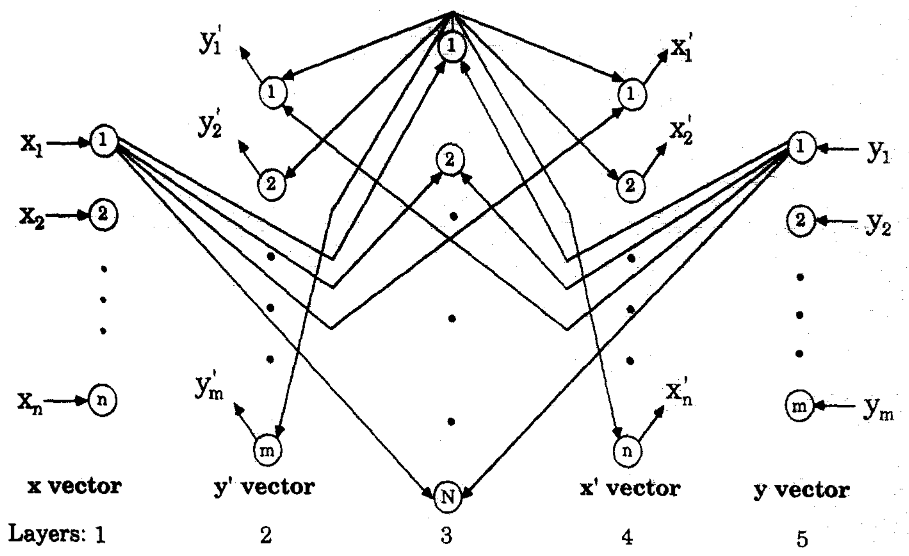

FIGURE 1. Topology of the counterpropagation network.

the restriction that x and y be on the unit sphere has been removed]. After each x is selected, $\mathbf{y} = \phi(\mathbf{x})$ or $\mathbf{y} = \phi(\mathbf{x}) + \mathbf{n}$ (where $\mathbf{n}$ is a relatively small stationary noise process) is determined and both x and y are input to the network. Thus, counterpropagation networks are trained via *supervised training* (as opposed to *graded training* or *self-organization*—the other two categories of neural network training—see Hecht-Nielsen, 1988a). The equations of the network and the network's operation during and after training are now briefly described. See Hecht-Nielsen (1987a, 1987b) for further details.

During training the transfer function equations for the Kohonen layer are:

$$z_i = \begin{cases} 1 & \text{if } i \text{ is the smallest integer for which} \\ & \|\mathbf{w}_i^{old} - \mathbf{x}\| \le \|\mathbf{w}_j^{old} - \mathbf{x}\| \text{ for all } j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \alpha(\mathbf{x} - \mathbf{w}_i^{old})z_i. \quad (2)$$

Thus, as shown by Equation 1, each time an x vector is submitted to the fanout layer, each processing element of the Kohonen layer receives all of its components. A

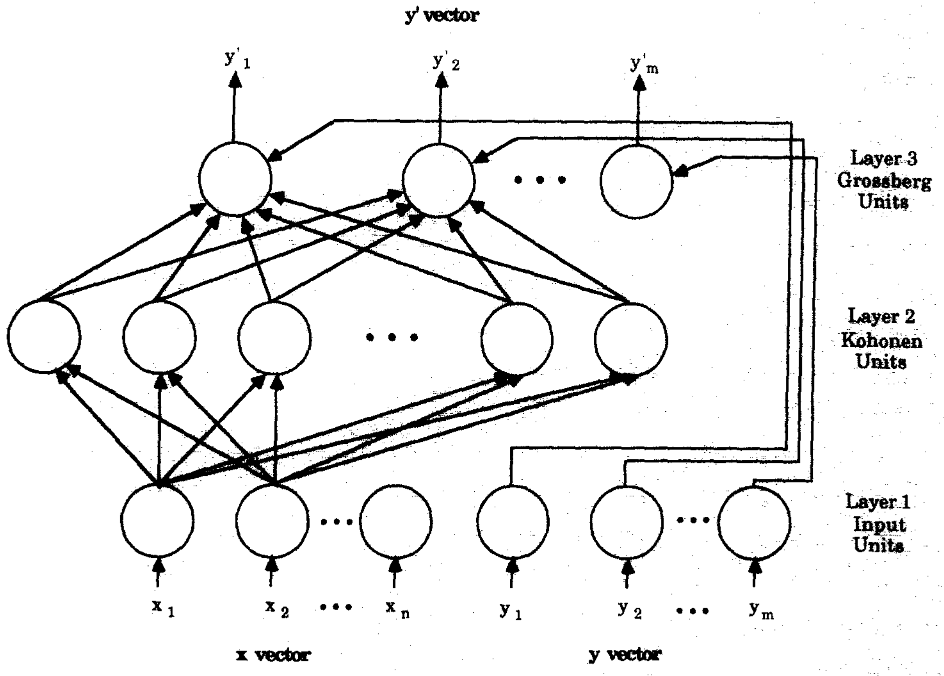

FIGURE 2. The forward-only counterpropagation network.

competition is then held among the units of the Kohonen layer (the mechanism for mediating this competition is not specified here; several different approaches, such as lateral inhibition and the use of a competition decision processing element work satisfactorily) and the one whose weight vector $\mathbf{w}_i$ is closest to $\mathbf{x}$ wins. This unit's output signal z_i is then set to 1 and the other processing element's output signals are set to 0. Ties are broken on the basis of the unit with the smallest index. Once the winner is determined it adjusts its weight vector $\mathbf{w}_i$ in accordance with Equation 2 by moving it to a new position $\mathbf{w}_i^{new}$ a fraction α of the way along the line segment extending from its present position $\mathbf{w}_i^{old}$ to $\mathbf{x}$. At the beginning of training, α usually starts at a value between 0.5 and 0.8. As training progresses, this value is lowered to 0.1 or less.

After a large number of inputs, the $\mathbf{w}_i$ vectors arrange themselves in $\mathbf{R}^n$ in such a way that they are approximately equiprobable in a nearest neighbor sense with respect to $\mathbf{x}$ vectors drawn from $\mathbf{R}^n$ in accordance with the probability density function ρ. In other words, given any i, $1 \leq i \leq N$, and given an $\mathbf{x}$ vector drawn from $\mathbf{R}^n$ in accordance with ρ, the probability that $\mathbf{x}$ is closest to $\mathbf{w}_i$ is approximately $1/N$. However, as discussed in Hecht-Nielsen (1987a, 1987b) the learning equation given above must often be modified for this equiprobable outcome to be obtained. The *conscience* and *noise addition* methods discussed in Hecht-Nielsen (1987a) can almost always overcome any problems that arise.

Once the Kohonen layer has stabilized (in other words, the $\mathbf{w}_i$ vectors are frozen after reaching equiprobability) the Grossberg layer begins to learn the correct $\mathbf{y}$ output signal for each weight vector $\mathbf{w}_i$ of the Kohonen layer. The equations of this layer are:

$$y'_k = \sum_{i=1}^{N} u_{ki}^{old} z_i \tag{3}$$

$$u_{ki}^{new} = u_{ki}^{old} + a(-u_{ki}^{old} + y_k)z_i \tag{4}$$

where $0 < a < 1$ and where y_k is the kth component of $\mathbf{y}$ (the desired output vector for the input $\mathbf{x}$). Equation 3 says that the output of the kth processing element of the Grossberg layer is equal to the sum of each Kohonen unit's output signal z_i times its input weight u_{ki}. Since (at least during training) one of the z_i is 1 and the others are 0, this sum has only one non-zero term. Equation 4 causes the weight u_{ki} associated with this single non-zero input (say z_i) to become approximately equal to the average of the kth component y_k of the $\mathbf{y}$ vectors that are entered when this particular Kohonen unit wins the competition on layer 2. This can easily be shown by integrating the closed-form solution to Equation 4 by parts and using the assumption that the y_k values have a well-defined average that remains approximately constant over any large number of trials in which this particular Kohonen unit wins the layer 2 competition (which is true for all continuous mappings where sufficient Kohonen units are used). Assuming that the mapping ϕ is approximately locally linearizable (i.e., it is well approximated by a linear function between nearby $\mathbf{w}_i$ win regions), then this average will approximate the kth component of $\phi(\mathbf{w}_i)$. In other words, after equilibration of the entire network, the output vector $\mathbf{y}'$ of the Grossberg layer will be approximately $\phi(\mathbf{w}_i)$, where i is the index of the winning processing element on the Kohonen layer (see Hecht-Nielsen (1987a) for further discussion of this matter). Thus, after equilibration, the network functions essentially as a lookup table with N entries; approximately $(\mathbf{w}_1, \phi(\mathbf{w}_1))$, $(\mathbf{w}_2, \phi(\mathbf{w}_2)), \ldots, (\mathbf{w}_N, \phi(\mathbf{w}_N))$. Presented with an input vector $\mathbf{x}$ the network finds the $\mathbf{w}_i$ vector that is closest to $\mathbf{x}$ and then emits the $\mathbf{y}'$ value associated with this weight vector. At the beginning of training, a is set to 0.1. This is maintained until well after the Kohonen layer stabilizes, whereupon a is gradually lowered to 0.

During normal operation (in other words, after training has ended and the learning law equations have been turned off) the counterpropagation network can be operated in an *interpolation* mode. In this mode more than one processing element (PE) can win the competition on the Kohonen layer (in other words, those with the second closest, third closest, etc. weight vectors). These winning PEs split the unit output signal that formerly was concentrated in the single winning processing elements output signal. The split is ordinarily accomplished using a criterion that the units with their weight vectors closest to $\mathbf{x}$ get the largest outputs. The sum of the z_i remains equal to 1. Non-winning Kohonen units still have zero output signals. The win criterion can be set up based upon distance measurement between the input and the weights (which might allow different numbers of winners on different occasions) or on the basis that a fixed number of processing elements (the closest 3, for example) get to win. We shall assume that we are dealing with this latter case. The exact partitioning scheme used to divide up the unit z output signal is problem dependent. This is an area where careful tweaking can greatly increase performance.

The result of employing interpolation is that the output vector $\mathbf{y}$ becomes a blend (a convex combination) of the fixed table lookups associated with the winning Kohonen units. For ϕ mappings that are locally linearizable (as defined above) this leads to significantly improved mapping approximation accuracy. In the sequel both the single winner version and the interpolation version of the forward-only counterpropagation network shall be used.

Finally, it is worth noting that the counterpropagation network, unlike other networks such as adaptive resonance (Carpenter & Grossberg, 1987) and backpropagation, is so simple in its structure and function that it can easily be re-expressed in the form of a statistics algorithm. Naturally, an advantage of the neural network formulation is that it explicitly identifies opportunities for parallel implementation.

3. PATTERN RECOGNITION

Most pattern recognition problems involve the classification of an unknown n-dimensional feature vector $\mathbf{x}$ into one of M classes. In many such problems the geometry of each class set (set of feature vectors belonging to that class) is reasonably well behaved (i.e., the set boundaries are reasonably smooth and simple, although the class sets may overlap). In these cases the counterpropagation network can function much like a Bayes classifier.

The basic idea is to equip the Grossberg layer of the network with M processing elements—each representing one of the M pattern classes. During training, as each feature vector $\mathbf{x}$ is entered into the fanout units of layer 1 of the network a "class vector" is entered into the Grossberg layer. This class vector has a 1 in the component corresponding to the class to which the feature vector pattern belongs and 0s in all of the other $M - 1$ components. Thus, the network learns the mapping from feature vector to class number—which is the essence of the pattern recognition function.

If interpolation is used the classifier can function approximately as a multiclass Bayes classifier (see Devijver & Kittler (1982) and Duda & Hart (1973) for discussions of pattern classification theory). For example, Figure 3 shows a situation in which there are three pattern classes. The class set of class 1 is a solid torus that is interlinked with (but does not intersect) the class 2 class set, which is a figure-8 shaped region. Class 3's class set is the rectangular region that intersects both the class 1 and class 2 class regions. We assume that a counterpropagation network with hundreds of Kohonen units has been trained on data from these classes. In this example the feature vectors are assumed to be uniformly likely to be selected from each volume element of each of the three class sets. To illustrate how the network will function in normal operation, consider the $\mathbf{y}$ outputs that the network will provide when test feature vectors A, B, and C are entered (see Figure 3). When A is entered, the Kohonen units compete with one another, and those nearest the point A win the competition. For Kohonen weight vectors in this region of class 2, the second Grossberg processing element's output (y_2) will be equal to 1. The other two processing element outputs (y_1 and y_3) will both be zero. (There are three Grossberg units in this network because there are three classes in this example.)

When point B (which is in the intersection of classes 1 and 3) is inserted into the network, the output of the Grossberg layer will be approximately $y_1 = 0.5$, $y_2 = 0.5$, and $y_3 = 0$. For point C (which is in the intersection of classes 2 and 3), the Grossberg layer outputs will be approximately $y_1 = 0$, $y_2 = 0.5$, and $y_3 = 0.5$. The accuracy to which these desired outputs are achieved is dependent upon the number of Kohonen units used and the number of layer 2 winners. Without sufficient Kohonen units the class set geometries are not well approximated and without a sufficient number of layer

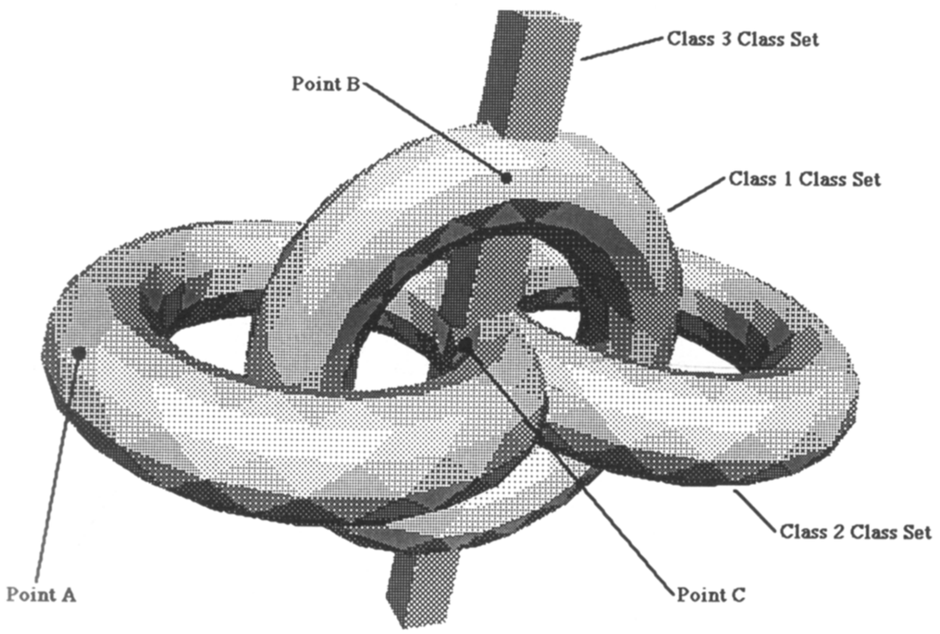

FIGURE 3. A three-class 3-dimensional pattern recognition feature space. Each class has the same constant probability density function value on a volumetric basis. Kohonen vectors have been spread uniformly throughout the classes by training. The use of counterpropagation in cases such as this can often save from one to three orders of magnitude in computational burden over k-nearest neighbor classification, with no increase in development expense or loss of performance.

2 winners the chances of undersampling one of the classes is increased. If we allow for a reasonable number of winners (say, 1 to 10), the weight vectors nearest point B will almost certainly have their Grossberg unit outputs approximately evenly divided between class 1 and class 3. Similarly, at point C, the outputs associated with the winning Kohonen weight vectors will be more or less evenly divided between class 2 and class 3. Note that just as in the Bayes classifer or the k-nearest neighbor classifier (see Duda & Hart (1973) for details), the output of the Grossberg layer unit for each class tends to be proportional to the local probability density function value of that class. In fact, these outputs are approximately *equal* to these probability density function values. This occurs because the sum of the outputs of the Kohonen layer units is 1 and because the outputs of the Grossberg layer units tend to also sum to 1. This happens because each of their weights become approximately equal to the average fraction of the time that points in the neighborhood of a particular Kohonen unit belong to a particular Grossberg unit's class (remember the y_i input to each Kohonen unit on each training trial is either 0 or 1, depending on whether the point $\mathbf{x}$ belongs to that particular class or not). Thus, the outputs of the Grossberg units are approximately equal to the probability density function values for their classes. Notice that unlike the k-nearest neighbor scheme, the counterpropagation network learns the approximate probability density function values at *each* weight vector point. Thus, an accurate density estimate can be obtained with only a few winners (or sometimes only one).

It is critical to notice that since counterpropagation uses the euclidean metric that this implies that it treats local distance differences on an isotropic basis. While the approach can clearly compensate for positional differences in probability density it ultimately will do a poor job of correctly representing the local probability density function values unless changes in coordinate value in one dimension are roughly as "important" as changes in all other dimensions (see Fukunaga & Hummels (1987) and references cited therein for further details). This means that (as with the k-nearest neighbor technique) it is essential that the feature coordinates be scaled correctly. Both linear (e.g., multiplicative) and nonlinear (e.g., logarithmic) scaling must be considered. Often it is necessary to scale multiple coordinates simultaneously via a nonlinear transformation. While this scaling requirement may seem difficult, it is often rather simple to find useable scales by a cut and fit approach based upon data from the probability density determination technique described in Section 5. The idea is to simply try different scales until the Kohonen weight vector (or feature vector) density is isotropic in all areas of the feature data. Another neurocomputing approach to this problem is to map the feature space manifold for a particular pattern recognition problem onto an n-dimensional rectangular grid using the Kohonen self-organizing map neural network Kohonen, 1988). This approach ensures that the images of the feature vectors will lie on a euclidean feature grid with uniform probability density. However, this approach must be used with caution since it can radically distort the geometry of the class sets. These comments apply to all uses of the counterpropagation network.

It can easily be seen from the example of Figure 3 that counterpropagation networks for pattern classification problems must usually be quite large. However, they are typically from one to three orders of magnitude smaller (and proportionally less demanding computationally) than a k-nearest neighbor system of the same level of performance. The counterpropagation approach is also expected to have both classification accuracy and computational economy advantages over both the edited k-nearest neighbor and nearest prototype classifier approaches, although this has not been tested yet. Thus, since the k-nearest neighbor approach and the counterpropagation approach are essentially the only purely data-driven non-parametric approaches that can achieve near-Bayesian performance in high-dimensional feature spaces this makes counterpropagation the technique of choice wherever a totally model-independent approach is needed (note: other traditional non-parametric approaches such as the Parzen window are computationally infeasible in spaces with more than about 10 dimensions). However, this is not to say that counterpropagation is the only answer for all pattern classification problems. For problems with relatively simple structure (or at least functionally appropriate structure) mapping neural networks such as backpropagation can be used to give high classification accuracy with a much smaller network by discovering effective feature sets.

Given the simplicity and ease of use of counterpropagation in comparison to other techniques requiring pattern structure analysis and algorithm development, the savings in development costs and development time that can be gained with its use are potentially significant. Since neurocomputers are relatively inexpensive, the increased implementation expense may often be acceptable.

Figure 4 illustrates another example of how the counterpropagation network can be used for pattern recognition. The problem in this example involves guiding a robot arm to a gripper fixture on a satellite. The robot arm is assumed to be attached to a repair hanger on the upcoming NASA space station. The arm has a camera mounted near its end effector, and this camera has been approximately boresighted on the satellite's gripper fixture by some other system. The gripper fixture is shaped like the letter "I." It is painted black, and is surrounded by a field that is painted white. The robot arm camera sees a scene similar to that shown in the upper left corner of Figure 4. The problem

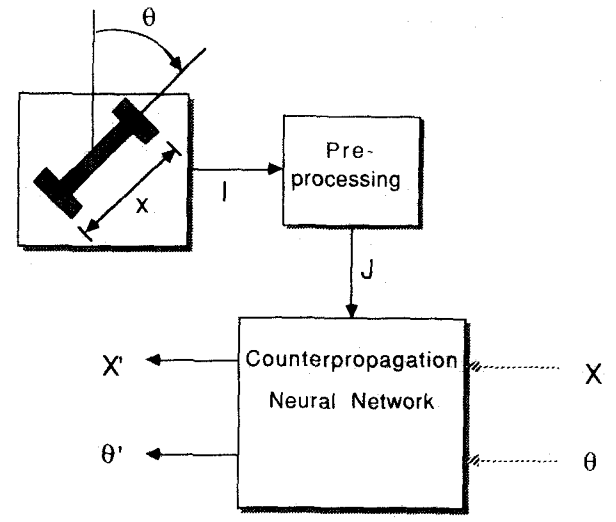

FIGURE 4. A counterpropagation network used to translate an image directly into pattern measurements x and θ.

is to determine the orientation angle θ and the apparent length x of the gripper fixture. This will allow the robot arm to extend and rotate its end effector to the proper distance and angle in order to grasp the gripper fixture. While this problem could clearly be solved using any of a number of different pattern recognition techniques, the goal was to see how quickly and easily it could be solved using neural networks. The result was that the angle measurement portion of the problem was solved (to within 3° of accuracy) using a counterpropagation network approach in approximately 60 hours of effort (including all of the following: gripper fixture model construction, lighting and TV camera setup, image frame grabber software development, user interface software development, image preprocessing and neural network control software development, counterpropagation network training, debugging, and final test). The approach used is described below. This work was carried out by James Freeman of Ford Aerospace in Houston, Texas and was funded by the Intelligent Systems Branch of the NASA Johnson Space Center.

The image of the gripper fixture was 32×32 pixels in size. Each pixel had an 8-bit unsigned grey-scale value. The first step was to threshold the image at a gray-scale level of 127 and assign a value of 0 to those pixels with grey scale values at or above this level and 1 to those with lower grey-scale values. This formed a binary image vector having 1,024 components. This vector was entered directly into a counterpropagation network with 12 Kohonen units and 2 Grossberg units. For the purposes of the initial experiment, all of the images were taken from same range (i.e., $x = constant$). As each image was entered, the correct θ value was supplied to the θ' output unit of the Grossberg layer; the other Grossberg layer output unit was used for x' and was not used in these experiments. The result was that, after a period of training, an arbitrary gripper image could be entered, and the network would identify the gripper fixture's orientation θ to within $\pm 3°$ of accuracy. This network used interpolation with two winners to achieve this result. The primary limitation to accuracy in this system was the win partitioning mechanism. In the form of interpolation used in these experiments there were *always* two winners, even when one of them was almost a perfect match for the input image. The second winning Kohonen unit in this situation was not given a zero output signal and so this caused an error. A different partitioning algorithm is expected to increase accuracy beyond $\pm 3°$. However, since the robot end effector orientation does not have to be even this accurate to successfully grip the fixture, the robot arm problem has already been solved, even with this suboptimal partitioning approach.

In summary, counterpropogation networks can be used for pattern recognition situations in which a quasi-Baysian classifier is desired or where template matching and template interpolation is desired.

4. FUNCTION APPROXIMATION

Because the counterpropagation network merely functions as a look-up table with equiprobable entries, there are very few functions for which the counterpropagation network will form a better approximation than a backpropagation network of a comparable size. However, examples can be developed for which this is true. For example, it is known that the backpropagation network can only accurately implement functions that can be approximated by multi-layer sigmoided linear combines (see Rumelhart & McClelland (1986) for details). It is possible to construct examples of functions that badly violate this constraint. However, such examples are typically contrived, and do not often appear in real-world problems. The advantage of the counterpropagation network for function approximation is that, regardless of the form of the function, given a sufficiently large network, an arbitrarily good approximation can be achieved.

Another area of function approximation where counterpropagation networks can be useful is in "rapid prototyping." This capability typically comes into play when developing a complicated information processing system that includes one or more mapping neural networks. The idea is to use the counterpropagation network module as an interim system component during development. At the end of development, after all of the other system components and the interactions between the various components have been tested and integrated, the counterpropagation network can be replaced with a more capable mapping network (such as backpropagation), which is essentially "plug-compatible" with the counterpropagation network. The advantage of this approach is that the counterpropagation network can usually be trained one to two orders of magnitude more quickly than a backpropagation network. During a development program, this might translate into hundreds or even thousands of hours of time savings. At the end of the development cycle, the counterpropagation network can be replaced with a more efficient mapping neural network that is then trained once.

5. STATISTICAL ANALYSIS

In traditional statistical analysis, the goal is to be able to answer questions that are posed about a data set. For example, in a database concerning automobile warranty repairs, we might want to know the probability that a front wheel bearing will go out on a 1984 Starshine convertible during its first 25,000 miles of operation as a function of the owner's geographical latitude. To answer this question, we build a statistical failure model, fit it to the data, and then answer the question. Statistical inferences of this sort, first of all, require us to pose a question, and second, to build a good statistical model to fit to the data.

Unlike this question-model-fitting approach, neurocomputing can, in certain cases, statistically analyze the structure of data without any questions being asked in advance. An example of this capability (which goes beyond the non-parametric statistical techniques typically encountered today) is presented below.

First of all, the Grossberg layer of the network is modified as follows:

$$y'_0 = \begin{cases} \sum_{i=1}^{N} z_i & \text{if } \sum_{i=1}^{N} z_i > 0 \\ 1 & \text{otherwise} \end{cases}$$

$$y''_k = \left(\sum_{k=1}^{N} u_{ki}^{old} z_i\right)/y'_0$$

$$\mathbf{y}' = P(\mathbf{y}'', S) - P(\mathbf{x}, S)$$

$$u_{ki}^{new} = u_{ki}^{old} + a(-u_{ki}^{old} + x_k)z_i$$

where $P(\mathbf{v}, S)$ is the projection of the vector $\mathbf{v}$ onto the subspace S ($S \subseteq \mathbf{R}^n$). The network is trained on data vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$, where these example vectors are assumed to be chosen in accordance with the fixed probability density function ρ. These vectors are entered into the network as input to the fanout units of layer 1. However, they are also entered into the $\mathbf{y}$ vector input units of layer 1 (see Figure 2). As indicated in the equations above, the Grossberg layer is further modified to include a 0th processing element (which is actually implemented as a separate slab and whose output is available to the usual Grossberg units). The output of this 0th unit is the sum of the output signals of the Kohonen layer processing elements.

During training, the Kohonen layer operates as described in Section 2. However, during normal operation, each of the Kohonen units is set up with a common threshold Γ. During normal operation, the transfer function of the Kohonen unit is modified as follows:

$$z_i = \begin{cases} 1 & \text{if } \|P(\mathbf{w}_i, S) - P(\mathbf{x}, S)\| \le \Gamma \\ 0 & \text{otherwise} \end{cases}$$

The operation of this network is now described.

During training, there is one winner on the Kohonen layer, and the weight vectors spread out to form a set of equiprobable representation vectors for the data environment from which the $\mathbf{x}$ vectors are chosen. In other words, they model the probability density function ρ. Since the Grossberg layer units receive the same vector $\mathbf{x}$ during training that the Kohonen units receive, the weights learn an approximation to the identity mapping (although the $\mathbf{y}'$ output of the network is approximately zero). After the network has equilibrated and the weight vectors have become equiprobable, learning is turned off, and normal operation begins.

The purpose of this network is to allow us to discover the strongest relationships between the fields in the data without having to ask any questions. This is best explained by describing how the network is used during normal operation to explore the structure of the data. First of all, notice that when a vector $\mathbf{x}$ is entered into the network that all of the Kohonen processing elements whose weight vectors lie within Γ distance units of x after both are projected onto the selected subspace S win the competition on the Kohonen layer and have output signals equal to 1. When these signals reach the Grossberg layer, processing element number zero outputs the number of weight vectors that lie within Γ distance units of the vector $\mathbf{x}$. Let us call this value $D(\mathbf{x}, \Gamma, S)$. If there are a sufficient number N of Kohonen units, if $S = \mathbf{R}^n$, and if Γ is chosen appropriately, $D(\mathbf{x}, \Gamma, S)/N$ is approximately equal to the probability density function value $\rho(\mathbf{x})$ (the network functions much like a Parzen window; see Devijver &Kittler, 1982; Devroye & Gyorfi, 1985). The $\mathbf{y}'$ vector that is emitted is therefore essentially equal to the average of the winning weight vectors minus the $\mathbf{x}$ vector that was entered. Since the number density of the weight vectors is approximately proportional to the probability density function, we can therefore expect more weight vectors in the direction of increasing probability. Therefore, this vector represents a crude estimate of the gradient of the probability function. The beauty of this approach is that there is essentially no additional effort required to calculate this vector. As with pattern classification, the advantage of using counterpropagation for statistical calculations is a significant reduction in computational burden as well as providing a direct path to parallel implementation. A sketch of how the network (once trained) can be used for exploring the statistics of a particular data set is now provided.

The first step, as indicated above, is to use the $\mathbf{x}$ data vectors (records) to train the network. Following this, the next step is to locate any significant dependencies or relationships between the components (fields) of the data. This is done by sequentially entering each 2-dimensional subspace (determined by coordinate basis vector pairs σ_i and σ_j; $i, j \in \{1, 2, \ldots, n\}$ $i \neq j$) for S and for each such subspace checking out the values of the $D(\mathbf{w}_k, \Gamma, S)$, for all $k \in \{1, 2, \ldots, N\}$. If any of these values turns out to be particularly high then this means that there is a significant relationship between the two variables x_i and x_j at this point. When such a discovery is made we can print out a graph of the projections of the $\mathbf{w}_k$ vectors (or even of the original data records) on this plane and let the user evaluate it visually. Such a scatter plot graph will be called an "H–R diagram" in light of its resemblance to the data plot of stellar color and absolute magnitude originated by astronomers E. Hurtzprung and H. N. Russell in 1911 and 1913, respectively.

Notice that any relationships between more than two data fields will show up initially on the relevant H–R diagrams. Thus, if high D values show up on two or more H–R diagrams that share a coordinate then the higher dimensional subspace(s) should be investigated. One approach is to go to a known position of 2-dimensional high density and follow this density using the gradient $\mathbf{y}'$. By this means, all important relationships between the data fields can be found without asking any questions. An automated data analysis system based on this approach would require virtually no technical knowledge on the part of the user. For example, if the $\mathbf{x}$ vectors were shoe sales records, the system would immediately discover a strong relationship between customer height and shoe size, and display a plot of this to the user. It would then go to discover less obvious, and more valuable, relationships such as a strong correlation between women's shoe color and the month of the year and an unexpectedly strong dependency of tennis shoe style on foot width. This technique is clearly capable of discovering relationships and connections that might never be found using a query-based approach that requires users to ask the right questions.

6. DATA COMPRESSION

One of the currently popular techniques in statistical data compression is *vector quantization* (Gersho & Shoham, 1984). In vector quantization the goal is to find a set of disjoint regions $A_1, A_2, \ldots, A_{2^M}$ of $\mathbf{R}^n$, with

$$\bigcup_{k=1}^{2^M} A_k = \mathbf{R}^n,$$

such that a data vector $\mathbf{x}$, chosen at random with respect to a fixed probability density function ρ is equally likely to be in each region. The compression scheme then works like this: the region A_k containing an $\mathbf{x}$ vector to be transmitted is identified and the index k of this region is transmitted instead of $\mathbf{x}$ (this requires M bits of information); on the receiving end (where a table of the statistical centroids of the regions is available) the centroid of the region A_k is emitted as a replacement for $\mathbf{x}$. This scheme works rather well. Speech, imagery, and many other types of analog data have been successfully compressed using it. Naturally, the data that is reconstructed on the receiving end is somewhat corrupted, and so this approach is not applicable in those situations where perfect reconstruction is required. For speech and imagery, compression ratios of 10:1 to 100:1 have been achieved with usable reconstruction.

A counterpropagation network can be used to carry out vector quantization data compression. The simplest scheme is where $N = 2^M$ Kohonen units and M Grossberg units are used. Once the Kohonen units have stabilized the $\mathbf{w}_k$ vectors become the centers of the A_k regions. Once the Kohonen units have equilibrated the $\mathbf{y}$ vector that is submitted to the Grossberg layer is the M-bit code for the number of the Kohonen unit that has its weight vector closest to $\mathbf{x}$ (this is arranged via processing elements on a separate slab added to the basic network). Thus, given a data vector input $\mathbf{x}$, the network will emit the M-bit binary code for the region containing it. Thus, the counterpropagation network can implement vector quantization directly. This capability has been demonstrated for imagery by several groups.

An advantage of counterpropagation over typical digital signal processing implementations of vector quantization (in addition to the inherent parallelism of the neural network approach) is that the A_k regions usually turn out to be significantly more isotropic. In the typical digital signal processing implementation the regions are chosen so that their boundaries are parallel to the coordinate axes. While it is difficult to define these regions, they are easy to implement because of the fact that each of them can be defined in terms of n inequalities involving the coordinates of $\mathbf{x}$. By evaluating these inequalities in a sequential manner, one coordinate at a time, the total number of arithmetic operations can be kept very small. The problem with such rectangular regions is that, although they are equiprobable, they are highly anisotropic. The overall accuracy of the vector quantization scheme is determined by the distance between the actual $\mathbf{x}$ vector and the centroid of its A_k region—which is emitted in place of $\mathbf{x}$ on the receiving end. Rectangles in high dimensional spaces look sort of like porcupines. The distance from the center to each of the $2n$ faces is very small and the distance from the center to each of the 2^n vertices is very large. Rectangles are highly anisotropic. Further, most of the volume of a high-dimensional rectangle is located in its "spines" (corners). Thus, on the average, the center of a rectangle is a very poor approximation of a point within the rectangle. The polygonal "win regions" of the counterpropagation network often tend to be more spherical than rectangular, thus increasing accuracy.

Finally, counterpropagation networks can be used in a hierarchical manner to achieve even more compression than would be feasible with a single network. For example, let us say that we wanted to code data vectors **x** using $M = 64$ bits. This is not a particularly large transmission size, but to achieve a vector quantizer with 64 bits of output we would need approximately one Kohonen unit for every grain of sand on Earth. Obviously, there is a problem here with scaling up. However, we can build up to this in stages. For example, what if we started out by building a 16-bit quantizer. After this system stabilized we would then take the differences $\mathbf{w}_k - \mathbf{x}$ (the error vectors of the first quantizer) and feed them into another 16-bit quantizer as the input vectors. Similarly, as this network stabilized we would feed the error vectors from it into yet another network and finally the errors from this would feed a fourth 16-bit quantizer network. The final result is four 16-bit numbers, each coding sets of vectors that specialize in successively smaller errors. The nice part is that this system can be built with a total of only 262,144 Kohonen units and 64 Grossberg units. While it will probably not perform quite as well as a full 64-bit quantizer, its performance may not be too much less. This approach depends upon the assumption that the errors at one weight vector of a particular network are statistically about the same as the errors at another weight vector.

In conclusion, counterpropagation networks have been found to be useful in a number of areas; including pattern recognition, function approximation, statistical analysis, and data compression. The advantages of counterpropagation over other approaches are typically in the areas of development cost, computational savings, and the use of an explicitly parallelizable architecture.

REFERENCES

Carpenter, G., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing,* **37,** 54–115.

Devijver, A., & Kittler, J. (1982). *Pattern recognition: A statistical approach.* Englewood Cliffs, NJ: Prentice-Hall International.

Devroye, L., & Gyorfi, L. (1985). *Nonparametric density estimation.* New York: Wiley.

Duda, R., & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Fukunaga, K., & Hummels, D. M. (1987). Bayes error estimation using Parzen and k-NN procedures. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **PAMI-9,** 634–643.

Gersho, A., & Shoham, Y. (1984). Hierarchical vector quantization of speech with dynamic codebook allocation. *Proceedings of the IEEE ICASSP 1984* (10.9/1-4). New York: IEEE Press.

Grossberg, S. (1969). Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, I. *Journal of Mathematics and Mechanics,* **19,** 53–91.

Grossberg, S. (1971). Embedding fields: Underlying philosophy, mathematics and applications to psychology, physiology, and anatomy. *Journal of Cybernetics,* **1,** 28–50.

Grossberg, S. (1982). *Studies of mind and brain.* Boston: Reidel.

Hecht-Nielsen, R. (1988a). Neurocomputer applications. *Proceedings of the 1987 IEEE Asilomar Signals & Systems Conference.* New York: IEEE Press.

Hecht-Nielsen, R. (1988b). Neurocomputer applications. In R. Eckmiller & C. von der Malsburg (Eds.), *Neural computers* (pp. 445–453). Berlin: Springer-Verlag.

Hecht-Nielsen, R. (1988c). Neurocomputing and artificial intelligence. In R. Trappl (Ed.), *Future and impacts of artificial intelligence.* Berlin: Springer-Verlag (in press).

Hecht-Nielsen, R. (1987a). Counterpropagation networks. *Applied Optics,* **26,** 4979–4984.

Hecht-Nielsen, R. (1987b). Counterpropagation networks. *Proceedings 1987 IEEE International Conference on Neural Networks.* New York: IEEE Press.

Kohonen, T. (1988). *Self-organization and associative memory* (2nd ed.). New York: Springer-Verlag.

Rumelhart, D., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition* (Vols. I & II). Cambridge, MA: MIT Press.