An Introduction to Computing with Neural Nets

Richard P. Lippmann

Abstract

Artificial neural net models have been studied for many years in the hope of achieving human-like performance in the fields of speech and image recognition. These models are composed of many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets. Computational elements or nodes are connected via weights that are typically adapted during use to improve performance. There has been a recent resurgence in the field of artificial neural nets caused by new net topologies and algorithms, analog VLSI implementation techniques, and the belief that massive parallelism is essential for high performance speech and image recognition. This paper provides an introduction to the field of artificial neural nets by reviewing six important neural net models that can be used for pattern classification. These nets are highly parallel building blocks that illustrate neural- net components and design principles and can be used to construct more complex systems. In addition to describing these nets, a major emphasis is placed on exploring how some existing classification and clustering algorithms can be performed using simple neuron-like components. Single-layer nets can implement algorithms required by Gaussian maximum-likelihood classifiers and optimum minimum-error classifiers for binary patterns corrupted by noise. More generally, the decision regions required by any classification algorithm can be generated in a straightforward manner by three-layer feed-forward nets.

INTRODUCTION

Artificial neural net models or simply "neural nets" go by many names such as connectionist models, parallel distributed processing models, and neuromorphic systems. Whatever the name, all these models attempt to achieve good performance via dense interconnection of simple computational elements. In this respect, artificial neural net structure is based on our present understanding of biological nervous systems. Neural net models have greatest potential in areas such as speech and image recognition where many hypotheses are pursued in parallel, high computation rates are required, and the current best systems are far from equaling human performance. Instead of performing a program of instructions sequentially as in a von Neumann computer, neural net models explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights.

Computational elements or nodes used in neural net models are nonlinear, are typically analog, and may be slow compared to modern digital circuitry. The simplest node sums N weighted inputs and passes the result through a nonlinearity as shown in Fig. 1. The node is characterized by an internal threshold or offset θ and by the type of nonlinearity. Figure 1 illustrates three common types of nonlinearities; hard limiters, threshold logic elements, and sigmoidal nonlinearities. More complex nodes may include temporal integration or other types of time dependencies and more complex mathematical operations than summation.

Neural net models are specified by the net topology, node characteristics, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. Both design procedures and training rules are the topic of much current research.

The potential benefits of neural nets extend beyond the high computation rates provided by massive parallelism. Neural nets typically provide a greater degree of robustness or fault tolerance than von Neumann sequential computers because there are many more processing nodes, each with primarily local connections. Damage to a few nodes or links thus need not impair overall performance significantly. Most neural net algorithms also adapt connection weights in time to improve performance based on current results. Adaptation or learning is a major focus of neural net research. The ability to adapt and continue learning is essential in areas such as speech recognition where training data is limited and new talkers, new words, new dialects, new phrases, and new environments are continuously encountered. Adaptation also provides a degree of robustness by compensating for minor variabilities in characteristics of processing elements. Traditional statistical techniques are not adaptive but typically process all training data simultaneously before being used with new data. Neural net classifiers are also non-parametric and make weaker assumptions concerning the shapes of underlying distributions than traditional statistical classifiers. They may thus prove to be more robust when distributions are generated by nonlinear processes and are strongly non-Gaussian. Designing artificial neural nets to solve

0740-7467/87/0400-0004/$01.00 ©1987 IEEE

problems and studying real biological nets may also change the way we think about problems and lead to new insights and algorithmic improvements.

Work on artificial neural net models has a long history. Development of detailed mathematical models began more than 40 years ago with the work of McCulloch and Pitts [30], Hebb [17], Rosenblatt [39], Widrow [47] and others [38]. More recent work by Hopfield [18, 19, 20], Rumelhart and McClelland [40], Sejnowski [43], Feldman [9], Grossberg [15], and others has led to a new resurgence of the field. This new interest is due to the development of new net topologies and algorithms [18, 19, 20, 41, 9], new analog VLSI implementation techniques [31], and some intriguing demonstrations [43, 20] as well as by a growing fascination with the functioning of the human brain. Recent interest is also driven by the realization that human-like performance in the areas of speech and image recognition will require enormous amounts of processing. Neural nets provide one technique for obtaining the required processing capacity using large numbers of simple processing elements operating in parallel.

This paper provides an introduction to the field of neural nets by reviewing six important neural net models that can be used for pattern classification. These massively parallel nets are important building blocks which can be used to construct more complex systems. The main purpose of this review is to describe the purpose and design of each net in detail, to relate each net to existing pattern classification and clustering algorithms that are normally implemented on sequential von Neumann computers, and to illustrate design principles used to obtain parallelism using neural-like processing elements.

Neural net and traditional classifiers

Block diagrams of traditional and neural net classifiers are presented in Fig. 2. Both types of classifiers determine which of M classes is most representative of an unknown static input pattern containing N input elements. In a speech recognizer the inputs might be the output envelope values from a filter bank spectral analyzer sampled at one time instant and the classes might represent different vowels. In an image classifier the inputs might be the gray scale level of each pixel for a picture and the classes might represent different objects.

The traditional classifier in the top of Fig. 2 contains two stages. The first computes matching scores for each class and the second selects the class with the maximum score. Inputs to the first stage are symbols representing values of the N input elements. These symbols are entered sequentially and decoded from the external symbolic form into an internal representation useful for performing arithmetic and symbolic operations. An algorithm computes a matching score for each of the M classes which indicates how closely the input matches the exemplar pattern for each class. This exemplar pattern is that pattern which is most representative of each class. In many situations a probabilistic model is used to model the generation of input patterns from exemplars and the matching score represents the likelihood or probability that the input pattern was generated from each of the M possible exemplars. In those cases, strong assumptions are typically made concerning underlying distributions of the input elements. Parameters of distributions can then be estimated using a training data as shown in Fig. 2. Multivariate Gaussian distributions are often used leading to relatively simple algorithms for computing matching scores [7]. Matching scores are coded into symbolic representations and passed sequentially to the second stage of

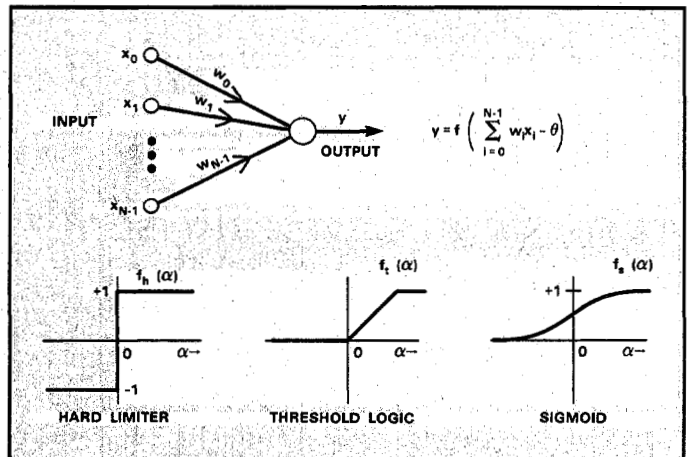

Figure 1. Computational element or node which forms a weighted sum of N inputs and passes the result through a nonlinearity. Three representative nonlinearities are shown.

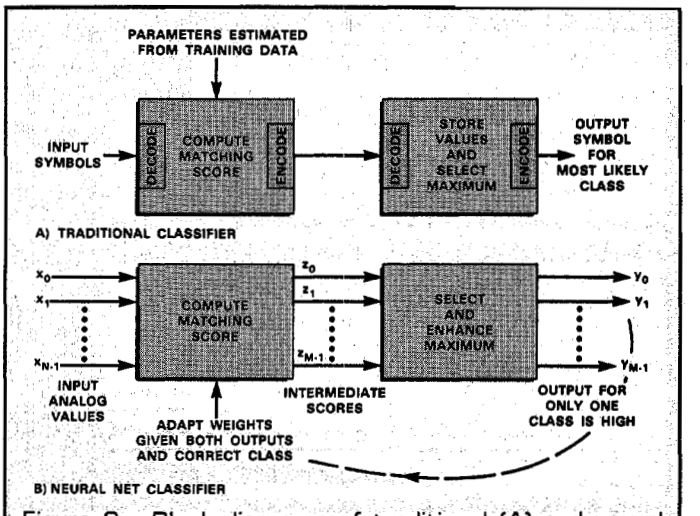

Figure 2. Block diagrams of traditional (A) and neural net (B) classifiers. Inputs and outputs of the traditional classifier are passed serially and internal computations are performed sequentially. In addition, parameters are typically estimated from training data and then held constant. Inputs and outputs to the neural net classifier are in parallel and internal computations are performed in parallel. Internal parameters or weights are typically adapted or trained during use using the output values and labels specifying the correct class.

the classifier. Here they are decoded and the class with the maximum score is selected. A symbol representing that class is then sent out to complete the classification task.

An adaptive neural net classifier is shown at the bottom of Fig. 2. Here input values are fed in parallel to the first stage via N input connections. Each connection carries an analog value which may take on two levels for binary inputs or may vary over a large range for continuous valued inputs. The first stage computes matching scores and outputs these scores in parallel to the next stage over M analog output lines. Here the maximum of these values is selected and enhanced. The second stage has one output for each of the M classes. After classification is complete, only that output corresponding to the most likely class will be on strongly or "high"; other outputs will be "low". Note that in this design, outputs exist for every class and that this multiplicity of outputs must be preserved in further processing stages as long as the classes are considered distinct. In the simplest classification system these output lines might go directly to lights with labels that specify class identities. In more complicated cases they may go to further stages of processing where inputs from other modalities or temporal dependencies are taken into consideration. If the correct class is provided, then this information and the classifier outputs can be fed back to the first stage of the classifier to adapt weights using a learning algorithm as shown in Fig. 2. Adaptation will make a correct response more likely for succeeding input patterns that are similar to the current pattern.

The parallel inputs required by neural net classifiers suggest that real-time hardware implementations should include special purpose pre-processors. One strategy for designing such processors is to build physiologically-based pre-processors modeled after human sensory systems. A pre-processor for image classification modeled after the retina and designed using analog VLSI circuitry is described in [31]. Pre-processor filter banks for speech recognition that are crude analogs of the cochlea have also been constructed [34, 29]. More recent physiologically-based pre-processor algorithms for speech recognition attempt to provide information similar to that available on the auditory nerve [11, 44, 27, 5]. Many of these algorithms include filter bank spectral analysis, automatic gain control, and processing which uses timing or synchrony information in addition to information from smoothed filter output envelopes.

Classifiers in Fig. 2 can perform three different tasks. First, as described above, they can identify which class best represents an input pattern, where it is assumed that inputs have been corrupted by noise or some other process. This is a classical decision theory problem. Second, the classifiers can be used as a content-addressable or associative memory, where the class exemplar is desired and the input pattern is used to determine which exemplar to produce. A content-addressable memory is useful when only part of an input pattern is available and the complete pattern is required, as in bibliographic retrieval of journal references from partial information. This normally requires the addition of a third stage in Fig. 2 to regenerate the exemplar for the most likely class. An additional stage is unnecessary for some neural nets such as the Hopfield net which are designed specifically as content-addressable memories. A third task these classifiers can perform is to vector quantize [28] or cluster [16, 7] the N inputs into M clusters. Vector quantizers are used in image and speech transmission systems to reduce the number of bits necessary to transmit analog data. In speech and image recognition applications they are used to compress the amount of data that must be processed without losing important information. In either application the number of clusters can be pre-specified or may be allowed to grow up to a limit determined by the number ber of nodes available in the first stage.

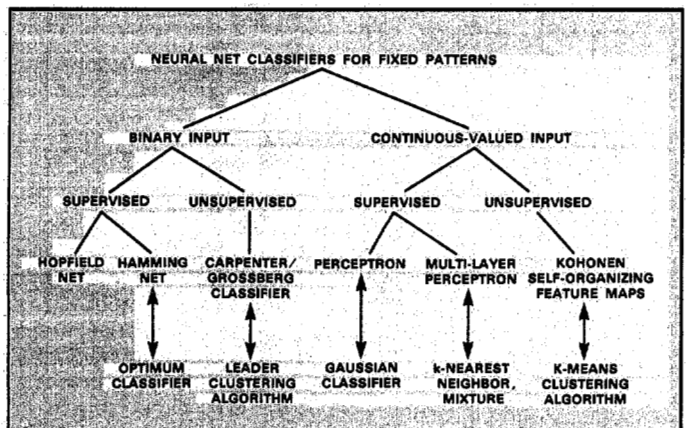

Figure 3. A taxonomy of six neural nets that can be used as classifiers. Classical algorithms which are most similar to the neural net models are listed along the bottom.

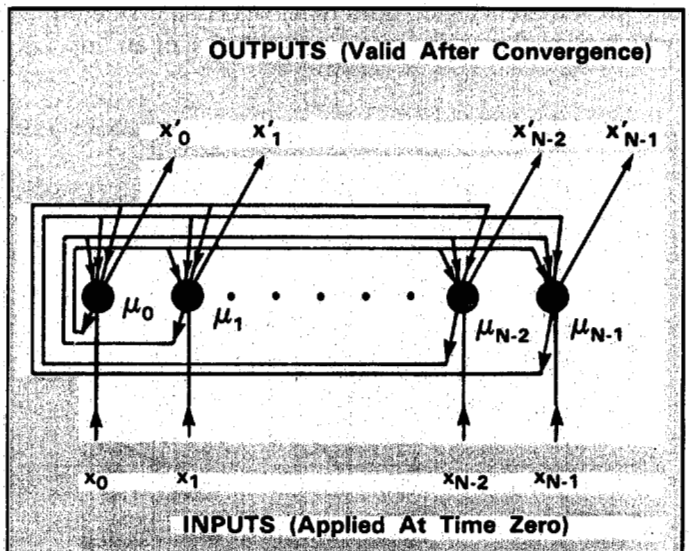

Figure 4. A Hopfield neural net that can be used as a content-addressable memory. An unknown binary input pattern is applied at time zero and the net then iterates until convergence when node outputs remain unchanged. The output is that pattern produced by node outputs after convergence.

A TAXONOMY OF NEURAL NETS

A taxonomy of six important neural nets that can be used for classification of static patterns is presented in Fig. 3. This taxonomy is first divided between nets with binary and continuous valued inputs. Below this, nets are divided between those trained with and without supervision. Nets trained with supervision such as the Hopfield net [18] and perceptrons [39] are used as associative memories or as classifiers. These nets are provided with side information or labels that specify the correct class for new input patterns during training. Most traditional statistical classifiers, such as Gaussian classifiers [7], are trained with supervision using labeled training data. Nets trained without supervision, such as the Kohonen's feature-map forming nets [22], are used as vector quantizers or to form clusters. No information concerning the correct class is provided to these nets during training. The classical K-means [7] and leader [16] clustering algorithms are trained without supervision. A further difference between nets, not indicated in Fig. 3, is whether adaptive training is supported. Although all the nets shown can be trained adaptively, the Hopfield net and the Hamming net are generally used with fixed weights.

The algorithms listed at the bottom of Fig. 3 are those classical algorithms which are most similar to or perform the same function as the corresponding neural net. In some cases a net implements a classical algorithm exactly. For example, the Hamming net [25] is a neural net implementation of the optimum classifier for binary patterns corrupted by random noise [10]. It can also be shown that the perceptron structure performs those calculations required by a Gaussian classifier [7] when weights and thresholds are selected appropriately. In other cases the neural net algorithms are different from the classical algorithms. For example, perceptrons trained with the perceptron convergence procedure [39] behave differently than Gaussian classifiers. Also, Kohonen's net [22] does not perform the iterative K-means training algorithm. Instead, each new pattern is presented only once and weights are modified after each presentation. The Kohonen net does, however, form a pre-specified number of clusters as in the K-means algorithm, where the K refers to the number of clusters formed.

THE HOPFIELD NET

The Hopfield net and two other nets in Fig. 3 are normally used with binary inputs. These nets are most appropriate when exact binary representations are possible as with black and white images where input elements are pixel values, or with ASCII text where input values could represent bits in the 8-bit ASCII representation of each character. These nets are less appropriate when input values are actually continuous, because a fundamental representation problem must be addressed to convert the analog quantities to binary values.

Hopfield rekindled interest in neural nets by his extensive work on different versions of the Hopfield net [18, 19, 20]. This net can be used as an associative memory or to solve optimization problems. One version of the original net [18] which can be used as a content addressable memory is described in this paper. This net, shown in Fig. 4, has N nodes containing hard limiting nonlinearities and binary inputs and outputs taking on the values +1 and −1. The output of each node is fed back to all other nodes via weights denoted t_{ij}. The operation of this net is described in Box 1. First, weights are set using the given recipe from exemplar patterns for all classes. Then an unknown pattern is imposed on the net at time zero by forcing the output of the net to match the unknown pattern. Following this initialization, the net iterates in discrete time steps using the given formula. The net is considered to have converged when outputs no longer change on successive iterations. The pattern specified by the node outputs after convergence is the net output.

Hopfield [18] and others [4] have proven that this net converges when the weights are symmetric ($t_{ij} = t_{ji}$) and

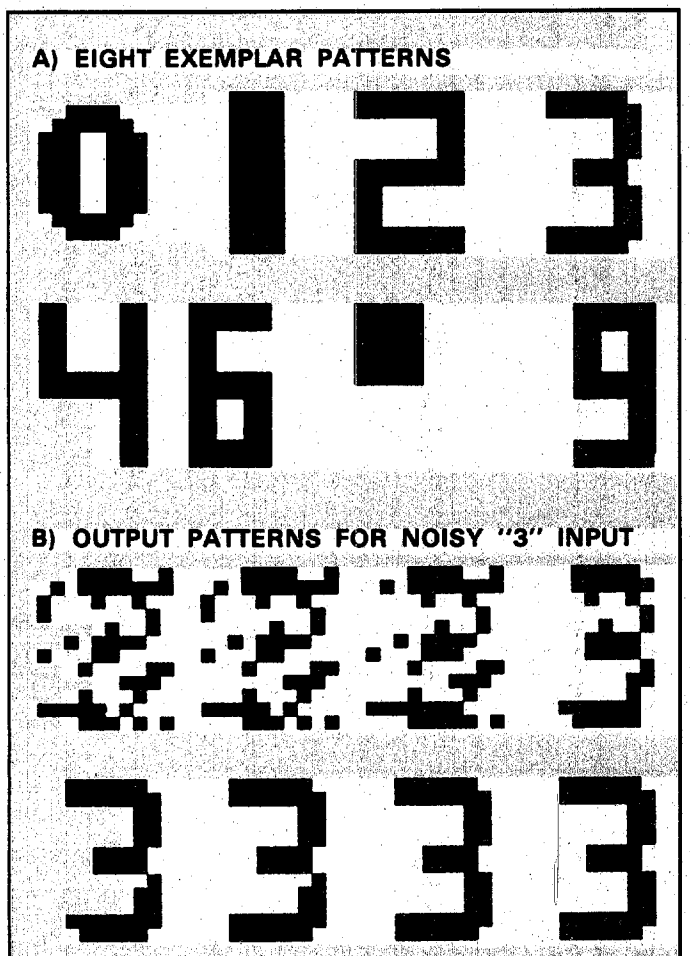

Figure 5. An example of the behavior of a Hopfield net when used as a content-addressable memory. A 120 node net was trained using the eight exemplars shown in (A). The pattern for the digit "3" was corrupted by randomly reversing each bit with a probability of .25, and then applied to the net at time zero. Outputs at time zero and after the first seven iterations are shown in (B).

node outputs are updated asynchronously using the equations in Box 1. Hopfield [19] also demonstrated that the net converges when graded nonlinearities similar to the sigmoid nonlinearity in Fig. 1 are used. When the Hopfield net is used as an associative memory, the net output after convergence is used directly as the complete restored memory. When the Hopfield net is used as a classifier, the output after convergence must be compared to the M exemplars to determine if it matches an exemplar exactly. If it does, the output is that class whose exemplar matched the output pattern. If it does not then a "no match" result occurs.

Box 1. Hopfield Net Algorithm

Step 1. Assign Connection Weights

$$t_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i^s x_j^s, & i \neq j \\ 0, & i = j, \ 0 \leq i, j \leq M-1 \end{cases}$$

In this Formula t_{ij} is the connection weight from node i to node j and x_i^s which can be +1 or −1 is element i of the exemplar for class s.

Step 2. Initialize with Unknown Input Pattern

$$\mu_i(0) = x_i, \quad 0 \leq i \leq N-1$$

In this Formula $\mu_i(t)$ is the output of node i at time t and x_i which can be +1 or −1 is element i of the input pattern.

Step 3. Iterate Until Convergence

$$\mu_j(t+1) = f_h\left[\sum_{i=0}^{N-1} t_{ij}\,\mu_i(t)\right], \quad 0 \leq j \leq M-1$$

The function f_h is the hard limiting nonlinearity from Fig. 1. The process is repeated until node outputs remain unchanged with further iterations. The node outputs then represent the exemplar pattern that best matches the unknown input.

Step 4. Repeat by Going to Step 2

The behavior of the Hopfield net is illustrated in Fig. 5. A Hopfield net with 120 nodes and thus 14,400 weights was trained to recall the eight exemplar patterns shown at the top of Fig. 5. These digit-like black and white patterns contain 120 pixels each and were hand crafted to provide good performance. Input elements to the net take on the value +1 for black pixels and −1 for white pixels. In the example presented, the pattern for the digit "3" was corrupted by randomly reversing each bit independently from +1 to −1 and vice versa with a probability of 0.25. This pattern was then applied to the net at time zero. Patterns produced at the output of the net on iterations zero to seven are presented at the bottom of Fig. 5. The corrupted input pattern is present unaltered at iteration zero. As the net iterates the output becomes more and more like the correct exemplar pattern until at iteration six the net has converged to the pattern for the digit three.

The Hopfield net has two major limitations when used as a content addressable memory. First, the number of patterns that can be stored and accurately recalled is severely limited. If too many patterns are stored, the net may converge to a novel spurious pattern different from all exemplar patterns. Such a spurious pattern will produce a "no match" output when the net is used as a classifier. Hopfield [18] showed that this occurs infrequently when exemplar patterns are generated randomly and the number of classes (M) is less than .15 times the number of input elements or nodes in the net (N). The number of classes is thus typically kept well below .15N. For example, a Hopfield net for only 10 classes might require more than 70 nodes and more than roughly 5,000 connection weights. A second limitation of the Hopfield net is that an exemplar pattern will be unstable if it shares many bits in common with another exemplar pattern. Here an exemplar is considered unstable if it is applied at time zero and the net converges to some other exemplar. This problem can be eliminated and performance can be improved by a number of orthogonalization procedures [14, 46].

THE HAMMING NET

The Hopfield net is often tested on problems where inputs are generated by selecting an exemplar and reversing bit values randomly and independently with a given probability [18, 12, 46]. This is a classic problem in communications theory that occurs when binary fixed-length signals are sent through a memoryless binary symmetric channel. The optimum minimum error classifier in this case calculates the Hamming distance to the exemplar for each class and selects that class with the minimum Hamming distance [10]. The Hamming distance is the number of bits in the input which do not match the corresponding exemplar bits. A net which will be called a Hamming net implements this algorithm using neural net components and is shown in Fig. 6.

The operation of the Hamming net is described in Box 2. Weights and thresholds are first set in the lower subnet such that the matching scores generated by the outputs of the middle nodes of Fig. 6 are equal to N minus the Hamming distances to the exemplar patterns. These matching scores will range from 0 to the number of elements in the input (N) and are highest for those nodes corresponding to classes with exemplars that best match the input. Thresholds and weights in the MAXNET subnet are fixed. All thresholds are set to zero and weights from each node to itself are 1. Weights between nodes are inhibitory with a value of $-\epsilon$ where $\epsilon < 1/M$.

After weights and thresholds have been set, a binary pattern with N elements is presented at the bottom of the Hamming net. It must be presented long enough to allow

the matching score outputs of the lower subnet to settle and initialize the output values of the MAXNET. The input is then removed and the MAXNET iterates until the output of only one node is positive. Classification is then complete and the selected class is that corresponding to the node with a positive output.

The behavior of the Hamming net is illustrated in Fig. 7.

Box 2. Hamming Net Algorithm

Step 1. Assign Connection Weights and Offsets

In the lower subnet:

$$w_{ij} = \frac{x_i^j}{2}, \qquad \theta_j = \frac{N}{2},$$
$$0 \le i \le N-1, \qquad 0 \le j \le M-1$$

In the upper subnet:

$$t_{kl} = \begin{cases} 1, & k = l \\ -\varepsilon, & k \ne l, \end{cases} \qquad \varepsilon < \frac{1}{M},$$
$$0 \le k, l \le M-1$$

In these equations w_{ij} is the connection weight from input i to node j in the lower subnet and θ is the threshold in that node. The connection weight from node k to node l in the upper subnet is t_{kl} and all thresholds in this subnet are zero. x_i^j is element i of exemplar j as in Box 1.

Step 2. Initialize with Unknown Input Pattern

$$\mu_j(0) = f_t\left(\sum_{i=0}^{N-1} w_{ij}x_i - \theta_j\right)$$
$$0 \le j \le M-1$$

In this equation $\mu_j(t)$ is the output of node j in the upper subnet at time t, x_i is element i of the input as in Box 1, and f_t is the threshold logic nonlinearity from Fig. 1. Here and below it is assumed that the maximum input to this nonlinearity never causes the output to saturate.

Step 3. Iterate Until Convergence

$$\mu_j(t+1) = f_t\left(\mu_j(t) - \varepsilon\sum_{k \ne j}\mu_k(t)\right)$$
$$0 \le j, k \le M-1$$

This process is repeated until convergence after which the output of only one node remains positive.

Step 4. Repeat by Going to Step 2

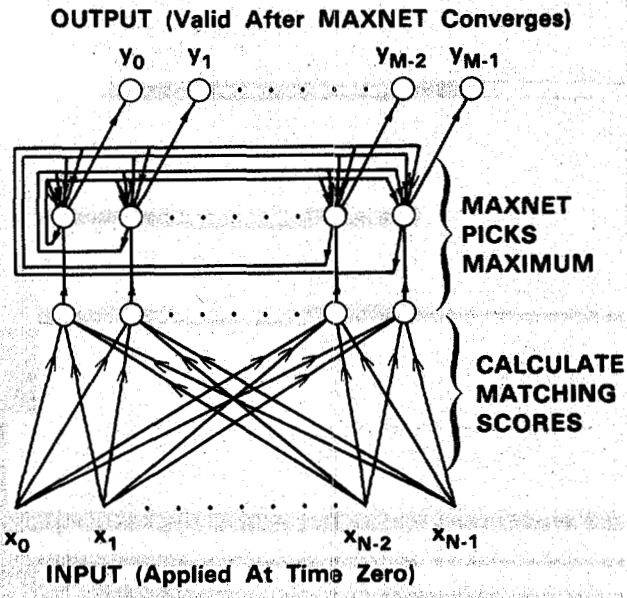

Figure 6. A feed-forward Hamming net maximum likelihood classifier for binary inputs corrupted by noise. The lower subnet calculates N minus the Hamming distance to M exemplar patterns. The upper net selects that node with the maximum output. All nodes use threshold-logic nonlinearities where it is assumed that the outputs of these nonlinearities never saturate.

The four plots in this figure show the outputs of nodes in a MAXNET with 100 nodes on iterations 0, 3, 6, and 9. These simulations were obtained using randomly selected exemplar patterns with 1000 elements each. The exemplar for class 50 was presented at time zero and then removed. The matching score at time zero is maximum (1000) for node 50 and has a random value near 500 for other nodes. After only 3 iterations, the outputs of all nodes except node 50 have been greatly reduced and after 9 iterations only the output for node 50 is non-zero. Simulations with different probabilities of reversing bits on input patterns and with different numbers of classes and elements in the input patterns have demonstrated that the MAXNET typically converges in less than 10 iterations in this application [25]. In addition, it can be proven that the MAXNET will always converge and find the node with the maximum value when $\varepsilon < 1/M$ [25].

The Hamming net has a number of obvious advantages over the Hopfield net. It implements the optimum minimum error classifier when bit errors are random and independent, and thus the performance of the Hopfield net must either be worse than or equivalent to that of the Hamming net in such situations. Comparisons between the two nets on problems such as character recognition, recognition of random patterns, and bibliographic retrieval have demonstrated this difference in performance [25]. The Hamming net also requires many fewer connections than the Hopfield net. For example, with 100 inputs and 10 classes the Hamming net requires

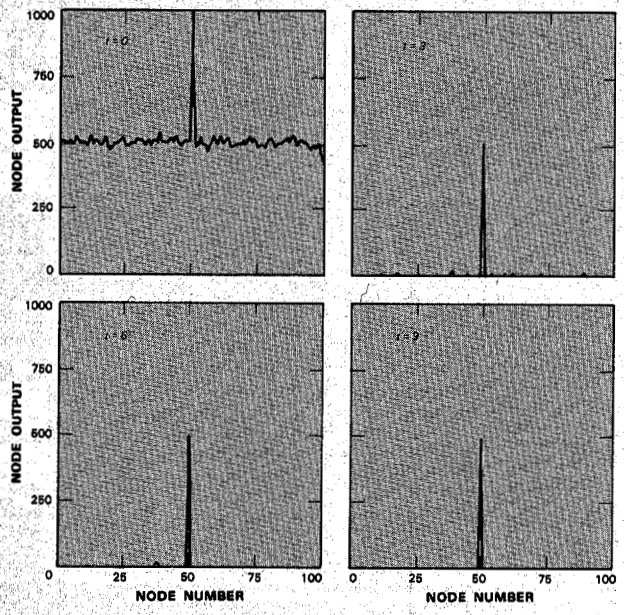

Figure 7. Node outputs for a Hamming net with 1,000 binary inputs and 100 output nodes or classes. Output values of all 100 nodes are presented at time zero and after 3, 6, and 9 iterations. The input was the exemplar pattern corresponding to output node 50.

only 1,100 connections while the Hopfield net requires almost 10,000. Furthermore, the difference in number of connections required increases as the number of inputs increases, because the number of connections in the Hopfield net grows as the square of the number of inputs while the number of connections in the Hamming net grows linearly. The Hamming net can also be modified to be a minimum error classifier when errors are generated by reversing input elements from +1 to −1 and from −1 to +1 asymmetrically with different probabilities [25] and when the values of specific input elements are unknown [2]. Finally, the Hamming net does not suffer from spurious output patterns which can produce a "no-match" result.

SELECTING OR ENHANCING THE MAXIMUM INPUT

The need to select or enhance the input with a maximum value occurs frequently in classification problems. Several different neural nets can perform this operation. The MAXNET described above uses heavy lateral inhibition similar to that used in other net designs where a maximum was desired [20, 22, 9]. These designs create a "winner-take-all" type of net whose design mimics the heavy use of lateral inhibition evident in the biological neural nets of the human brain [21]. Other techniques to pick a maximum are also possible [25]. One is illustrated in Fig. 8. This figure shows a comparator subnet which is described in [29]. It uses threshold logic nodes to pick the maximum of two inputs and then feeds this maximum value forward. This net is useful when the maximum value must be passed unaltered to the output. Comparator subnets can be layered into roughly $\log_2(M)$ layers to pick the maximum of M inputs. A net that uses these subnets to pick the maximum of 8 inputs is presented in Fig. 9.

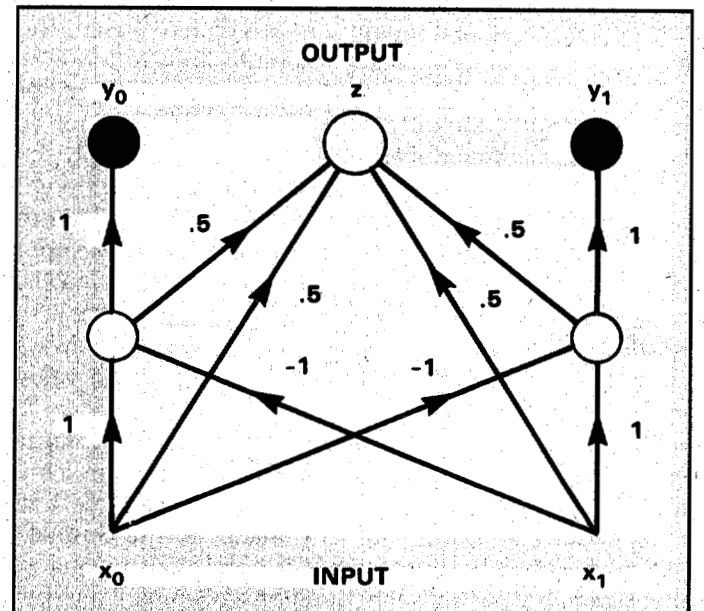

Figure 8. A comparator subnet that selects the maximum of two analog inputs. The output labeled z is the maximum value and the outputs labeled y_0 and y_1 indicate which input was maximum. Internal thresholds on threshold logic nodes (open circles) and hard limiting nodes (filled circles) are zero. Weights are as shown.

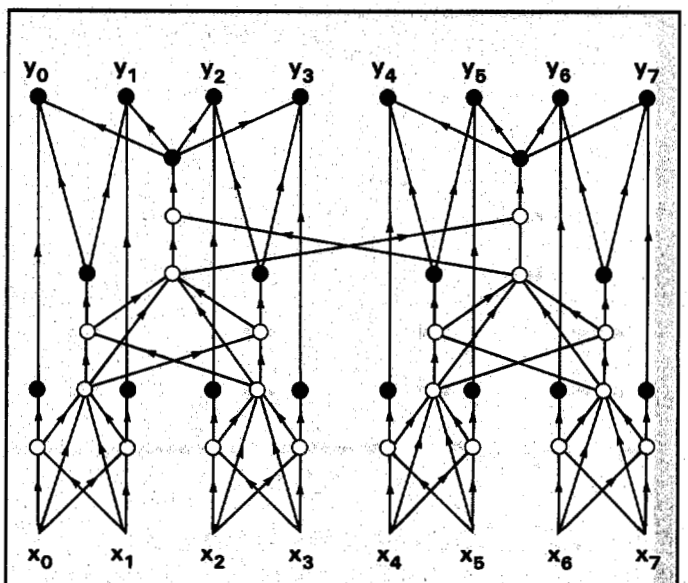

Figure 9. A feed-forward net that determines which of eight inputs is maximum using a binary tree and comparator subnets from Fig. 8. After an input vector is applied, only that output corresponding to the maximum input element will be high. Internal thresholds on threshold-logic nodes (open circles) and on hard limiting nodes (filled circles) are zero except for the output nodes. Thresholds in the output nodes are 2.5. Weights for the comparator subnets are as in Fig. 8 and all other weights are 1.

In some situations a maximum is not required and matching scores must instead be compared to a threshold. This can be done using an array of hard-limiting nodes with internal thresholds set to the desired threshold values. Outputs of these nodes will be −1 unless the inputs exceed the threshold values. Alternatively, thresholds could be set adaptively using a common inhibitory input fed to all nodes. This threshold could be ramped up or down until the output of only one node was positive.

THE CARPENTER/GROSSBERG CLASSIFIER

Carpenter and Grossberg [3], in the development of their Adaptive Resonance Theory have designed a net which forms clusters and is trained without supervision. This net implements a clustering algorithm that is very similar to the simple sequential leader clustering algorithm described in [16]. The leader algorithm selects the first input as the exemplar for the first cluster. The next input is compared to the first cluster exemplar. It "follows the leader" and is clustered with the first if the distance to the first is less than a threshold. Otherwise it is the exemplar for a new cluster. This process is repeated for all following inputs. The number of clusters thus grows with time and depends on both the threshold and the distance metric used to compare inputs to cluster exemplars.

The major components of a Carpenter/Grossberg classification net with three inputs and two output nodes is presented in Fig. 10. The structure of this net is similar to that of the Hamming net. Matching scores are computed using feed-forward connections and the maximum value is enhanced using lateral inhibition among the output nodes. This net differs from the Hamming net in that feedback connections are provided from the output nodes to the input nodes. Mechanisms are also provided to turn off that output node with a maximum value, and to compare exemplars to the input for the threshold test required by the leader algorithm. This net is completely described using nonlinear differential equations, includes extensive feedback, and has been shown to be stable [3]. In typical operation, the differential equations can be shown to implement the clustering algorithm presented in Box 3.

Box 3. Carpenter/Grossberg Net Algorithm

Step 1. Initialization

$$t_{ij}(0) = 1$$

$$b_{ij}(0) = \frac{1}{1+N}$$

$$0 \le i \le N-1,$$

$$0 \le j \le M-1$$

Set ρ, $\quad 0 \le \rho \le 1$

In these equations $b_{ij}(t)$ is the bottom up and $t_{ij}(t)$ is the top down connection weight between input node i and output node j at time t. These weights define the exemplar specified by output node j. The fraction ρ is the vigilance threshold which indicates how close an input must be to a stored exemplar to match.

Step 2. Apply New Input

Step 3. Compute Matching Scores

$$\mu_j = \sum_{i=0}^{N-1} b_{ij}(t)x_i \qquad 0 \le j \le M-1$$

In this equation μ_j is the output of output node j and x_i is element i of the input which can be 0 or 1.

Step 4. Select Best Matching Exemplar

$$\mu_j^* = \max_j \{\mu_j\}$$

This is performed using extensive lateral inhibition as in the maxnet.

Step 5. Vigilance Test

$$\|X\| = \sum_{i=0}^{N-1} x_i$$

$$\|T \cdot X\| = \sum_{i=0}^{N-1} t_{ij^*} x_i$$

is $\dfrac{\|T \cdot X\|}{\|X\|} > \rho$?

NO → GO TO STEP 6 YES → GO TO STEP 7

Step 6. Disable Best Matching Exemplar

The output of the best matching node selected in Step 4 is temporarily set to zero and no longer takes part in the maximization of Step 4. Then go to Step 3.

Step 7. Adapt Best Matching Exemplar

$$t_{ij^*}(t+1) = t_{ij^*}(t)x_i$$

$$b_{ij^*}(t+1) = \frac{t_{ij}^*(t)x_i}{.5 + \sum_{i=0}^{N-1} t_{ij^*}(t)x_i}$$

Step 8. Repeat by Going to Step 2

(First enable any nodes disabled in Step 6)

The algorithm presented in Box 3 assumes that "fast learning" is used as in the simulations presented in [3] and thus that elements of both inputs and stored exemplars take on only the values 0 and 1. The net is initialized by effectively setting all exemplars represented by connection weights to zero. In addition, a matching threshold called *vigilance* which ranges between 0.0 and 1.0 must be set. This threshold determines how close a new input pattern must be to a stored exemplar to be considered similar. A value near one requires a close match and smaller values accept a poorer match. New inputs are presented sequentially at the bottom of the net as in the Hamming net. After presentation, the input is compared to all stored exemplars in parallel as in the Hamming net to produce matching scores. The exemplar with the highest matching score is selected using lateral inhibition. It is then compared to the input by computing the ratio of the dot product of the input and the best matching exemplar (number of 1 bits in common) divided by the number of 1 bits in the input. If this ratio is greater than the vigilance threshold, then the input is considered to be similar to the best matching exemplar and that exemplar is updated by performing a logical AND operation between its bits and those in the input. If the ratio is less than the vigilance threshold, then the input is considered to be different from all exemplars and it is added as a new exemplar. Each additional new exemplar requires one node and 2N connections to compute matching scores.

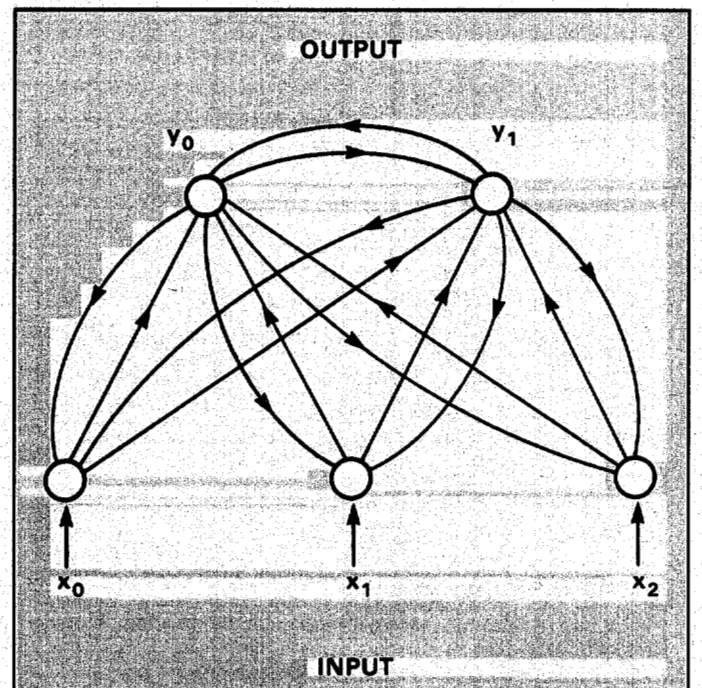

Figure 10. The major components of the Carpenter Grossberg Classification net. A binary input is presented at the bottom and when classification is complete only one output is high. Not shown are additional components required to perform the vigilance test and to disable the output node with the largest output.

The behavior of the Carpenter/Grossberg net is illustrated in Fig. 11. Here it is assumed that patterns to be recognized are the three patterns of the letters "*C*", "*E*", and "*F*" shown in the left side of this figure. These patterns have 64 pixels each that take on the value 1 when black and 0 when white. Results are presented when the vigilance threshold was set to 0.9. This forces separate exemplar patterns to be created for each letter.

The left side of Fig. 11 shows the input to the net on successive trials. The right side presents exemplar patterns formed after each pattern had been applied. In this example "C" was presented first followed by "E" followed by "F", etc. After the net is initialized and a "C" is applied, internal connection weights are altered to form an internal exemplar that is identical to the "C". After an "E" is then applied, a new "E" exemplar is added. Behavior is similar for a new "F" leading to three stored exemplars. If the vigilance threshold had been slightly lower, only two exemplars would have been present after the "F"; one for "F" and one for both "C" and "E" that would have been identical to "C" pattern. Now, when a noisy "F" is applied

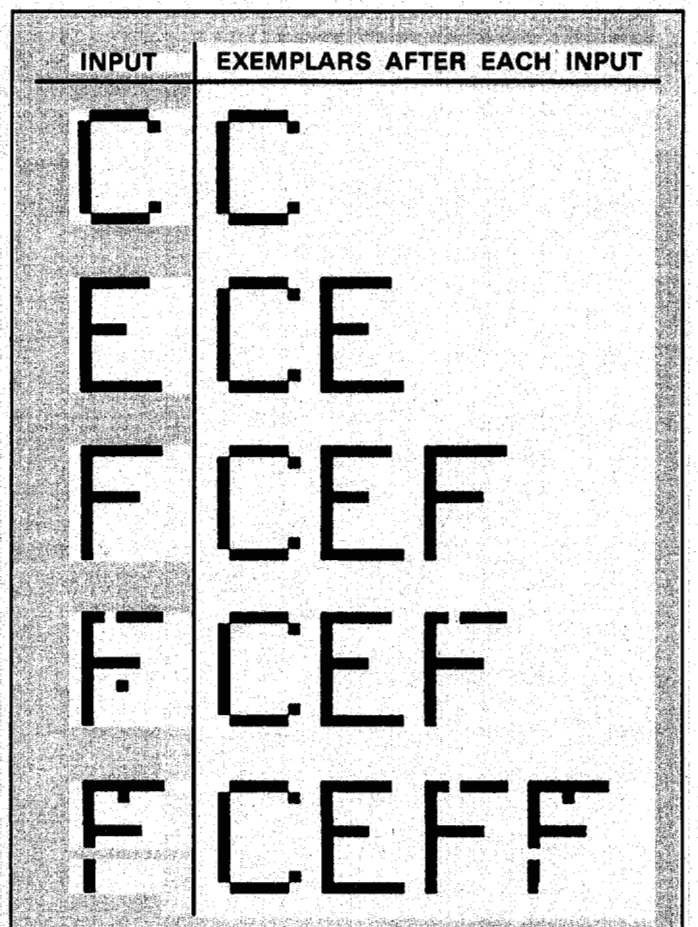

Figure 11. An example of the behavior of the Carpenter Grossberg net for letter patterns. Binary input patterns on the left were applied sequentially starting with the upper "C" pattern. Exemplars formed by top-down connection weights after each input was presented are shown at the right.

with a missing black pixel in the upper edge it is accepted as being similar to the "F" exemplar and degrades this exemplar due to the AND operation performed during updating. When another noisy "F" is applied again with only one black pixel missing, it is considered different from existing exemplars and a new noisy "F" exemplar is added. This will occur for further noisy "F" inputs leading to a growth of noisy "F" exemplars.

These results illustrate that the Carpenter/Grossberg algorithm can perform well with perfect input patterns but that even a small amount of noise can cause problems. With no noise, the vigilance threshold can be set such that the two patterns which are most similar are considered different. In noise, however, this level may be too high and the number of stored exemplars can rapidly grow until all available nodes are used up. Modifications are necessary to enhance the performance of this algorithm in noise. These could include adapting weights more slowly and changing the vigilance threshold during training and testing as suggested in [3].

SINGLE LAYER PERCEPTRON

The single layer perceptron [39] is the first of three nets from the taxonomy in Fig. 3 that can be used with both continuous valued and binary inputs. This simple net generated much interest when initially developed because of its ability to learn to recognize simple patterns. A perceptron that decides whether an input belongs to one of two classes (denoted A or B) is shown in the top of Fig. 12. The single node computes a weighted sum of the input elements, subtracts a threshold (θ) and passes the result through a hard limiting nonlinearity such that the output y is either +1 or −1. The decision rule is to respond class A if the output is +1 and class B if the output is −1. A useful technique for analyzing the behavior of nets such as the perceptron is to plot a map of the decision regions created in the multidimensional space spanned by the input variables. These decision regions specify which input values result in a class A and which result in a class B response. The perceptron forms two decision regions separated by a hyperplane. These regions are shown in the right side of Fig. 12 when there are only two inputs and the hyperplane is a line. In this case inputs above the boundary line lead to class A responses and inputs below the line lead to class B responses. As can be seen, the equation of the boundary line depends on the connection weights and the threshold.

Connection weights and the threshold in a perceptron can be fixed or adapted using a number of different algorithms. The original perceptron convergence procedure for adjusting weights was developed by Rosenblatt [39]. It is described in Box 4. First connection weights and the threshold value are initialized to small random non-zero values. Then a new input with N continuous valued elements is applied to the input and the output is computed as in Fig. 12. Connection weights are adapted only when an error occurs using the formula in step 4 of Box 4. This formula includes a gain term (η) that ranges from 0.0 to 1.0 and controls the adaptation rate. This gain term must be adjusted to satisfy the conflicting requirements of fast adaptation for real changes in the input distributions and averaging of past inputs to provide stable weight estimates.

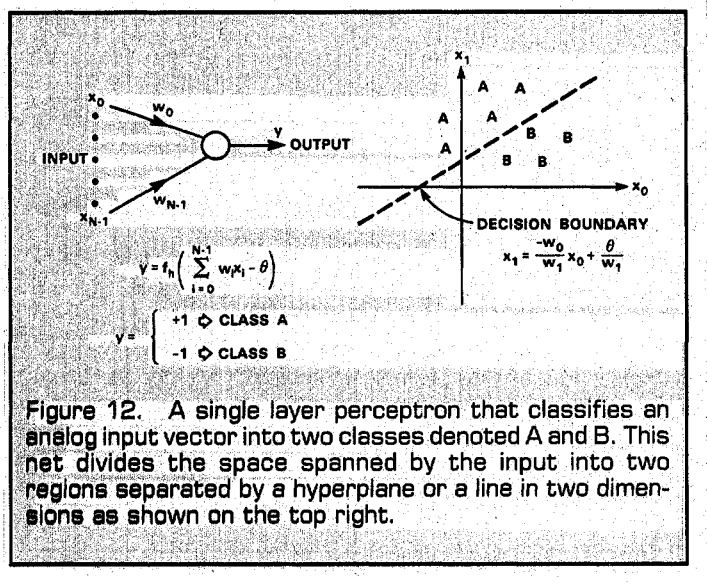

Figure 12. A single layer perceptron that classifies an analog input vector into two classes denoted A and B. This net divides the space spanned by the input into two regions separated by a hyperplane or a line in two dimensions as shown on the top right.

Box 4. The Perceptron Convergence Procedure

Step 1. Initialize Weights and Threshold

Set $w_i(0)$ $(0 \le i \le N-1)$ and θ to small random values. Here $w_i(t)$ is the weight from input i at time t and θ is the threshold in the output node.

Step 2. Present New Input and Desired Output

Present new continuous valued input $x_0, x_1 \ldots x_{N-1}$ along with the desired output $d(t)$.

Step 3. Calculate Actual Output

$$y(t) = f_h\left(\sum_{i=0}^{N-1} w_i(t)x_i(t) - \theta\right)$$

Step 4. Adapt Weights

$$w_i(t+1) = w_i(t) + \eta[d(t) - y(t)]x_i(t), \quad 0 \le i \le N-1$$

$$d(t) = \begin{cases} +1 \text{ if input from class A} \\ -1 \text{ if input from class B} \end{cases}$$

In these equations η is a positive gain fraction less than 1 and $d(t)$ is the desired correct output for the current input. Note that weights are unchanged if the correct decision is made by the net.

Step 5. Repeat by Going to Step 2

An example of the use of the perceptron convergence procedure is presented in Fig. 13. Samples from class A in this figure are represented by circles and samples from class B are represented by crosses. Samples from classes A and B were presented alternately until 80 inputs had been presented. The four lines show the four decision boundaries after weights had been adjusted following errors on trials 0, 2, 4, and 80. In this example the classes were well separated after only four trials and the gain term was .01.

Rosenblatt [39] proved that if the inputs presented from the two classes are separable (that is they fall on opposite sides of some hyperplane), then the perceptron convergence procedure converges and positions the decision hyperplane between those two classes. Such a hyperplane is illustrated in the upper right of Fig. 12. This decision boundary separates all samples from the A and B classes. One problem with the perceptron convergence procedure is that decision boundaries may oscillate continuously when inputs are not separable and distributions overlap. A modification to the perceptron convergence procedure can form the least mean square (LMS) solution in this case. This solution minimizes the mean square error between the desired output of a perceptron-like net and the actual output. The algorithm that forms the LMS solution is called the Widrow-Hoff or LMS algorithm [47, 48, 7].

The LMS algorithm is identical to the perceptron convergence procedure described in Box 4 except the hard limiting nonlinearity is made linear or replaced by a threshold-logic nonlinearity. Weights are thus corrected on every trial by an amount that depends on the difference between the desired and the actual input. A classifier that uses the LMS training algorithm could use desired outputs of 1 for class A and 0 for class B. During operation the input would then be assigned to class A only if the output was above 0.5.

The decision regions formed by perceptrons are similar to those formed by maximum likelihood Gaussian classifiers which assume inputs are uncorrelated and distributions for different classes differ only in mean values. This type of Gaussian classifier and the associated weighted Euclidean or straight Euclidean distance metric is often used in speech recognizers when there is limited training data and inputs have been orthogonalized by a suitable transformation [36]. Box 5 demonstrates how the weights and threshold in a perceptron can be selected such that the perceptron structure computes the difference between log likelihoods required by such a Gaussian classifier [7]. Perceptron-like structures can also be used to perform the linear computations required by a Karhunen Loeve transformation [36]. These computations can be used to transform a set of $N + K$ correlated Gaussian inputs into a reduced set of N uncorrelated inputs which can be used with the above Gaussian classifier.

It is straightforward to generalize the derivation of Box 5 to demonstrate how a Gaussian classifier for M classes can be constructed from M perceptron-like structures followed by a net that picks the maximum. The required net is identical in structure to the Hamming Net of Fig. 6. In this case, however, inputs are analog and the weights and node thresholds are calculated from terms II and III in likelihood equations similar to those for L_A in Box 5. It is likewise straightforward to generalize the Widrow-Hoff

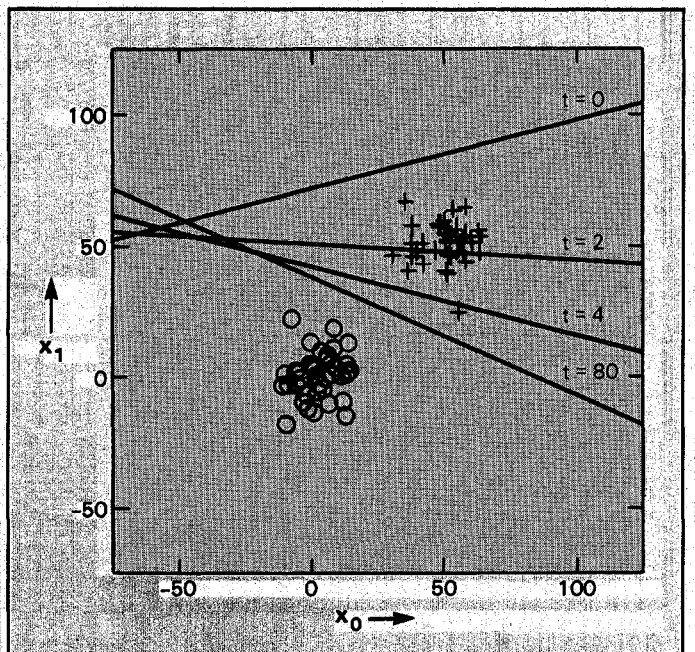

Figure 13. An example of the decision boundaries formed by the perceptron convergence procedure with two classes. Samples from class A are represented by circles and samples from class B by crosses. Lines represent decision boundaries after trials where errors occurred and weights were adapted.

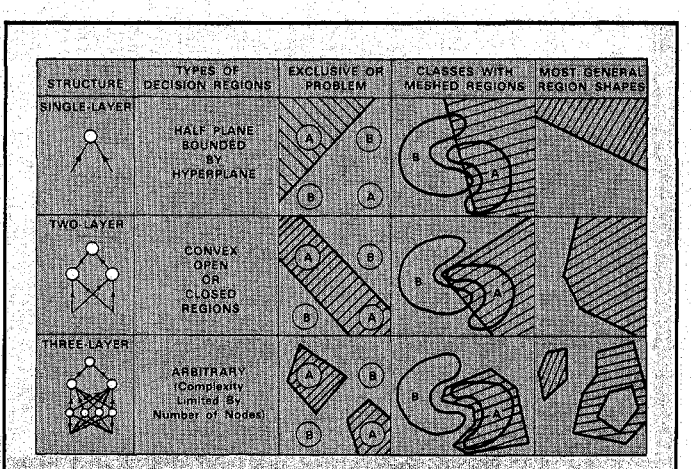

Figure 14. Types of decision regions that can be formed by single- and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A. Smooth closed contours bound input distributions for classes A and B. Nodes in all nets use hard limiting nonlinearities.

variant of the perceptron convergence procedure to apply for M classes. This requires a structure identical to the Hamming Net and a classification rule that selects the class corresponding to the node with the maximum output. During adaptation the desired output values can be set to 1 for the correct class and 0 for all others.

Box 5. A Gaussian Classifier Implemented Using the Perceptron Structure

If m_{Ai} and σ_{Ai}^2 are the mean and variance of input x_i when the input is from class A and M_{Bi} and σ_{Bi}^2 are the mean and variance of input x_i for class B and $\sigma_i^2 = \sigma_{Ai}^2 = \sigma_{Bi}^2$, then the likelihood values required by a maximum likelihood classifier are monotonically related to

$$L_A = -\sum_{i=0}^{N-1} \frac{(x_i - M_{Ai})^2}{\sigma_i^2}$$

$$= -\sum \frac{x_i^2}{\sigma_i^2} + 2\sum \frac{M_{Ai} x_i}{\sigma_i^2} - \sum \frac{M_{Ai}^2}{\sigma_i^2}$$

and

$$L_B = -\sum_{i=0}^{N-1} \frac{(x_i - M_{Bi})^2}{\sigma_i^2}$$

$$= -\sum \frac{x_i^2}{\sigma_i^2} + 2\sum \frac{M_{Bi} x_i}{\sigma_i^2} - \sum \frac{M_{Bi}^2}{\sigma_i^2},$$

Term I Term II Term III

A maximum likelihood classifier must calculate L_A and L_B and select the class with the highest likelihood. Since Term I in these equations is identical for L_A and L_B, it can be dropped. Term II is a product of the input times weights and can be calculated by a perception and Term III is a constant which can be obtained from the threshold in a perceptron node. A Gaussian classifier for two classes can thus be formed by using the perceptron of Fig. 12 to calculate $L_A - L_B$ by setting

$$w_i = \frac{2(M_{Ai} - M_{Bi})}{\sigma_i^2},$$

and

$$\theta = \sum_{i=0}^{N-1} \frac{M_{Ai}^2 - M_{Bi}^2}{\sigma_i^2}.$$

The perceptron structure can be used to implement either a Gaussian maximum likelihood classifier or classifiers which use the perceptron training algorithm or one of its variants. The choice depends on the application. The perceptron training algorithm makes no assumptions concerning the shape of underlying distributions but focuses on errors that occur where distributions overlap. It may thus be more robust than classical techniques and work well when inputs are generated by nonlinear processes and are heavily skewed and non-Gaussian. The Gaussian classifier makes strong assumptions concerning underlying distributions and is more appropriate when distributions are known and match the Gaussian assumption. The adaptation algorithm defined by the perceptron convergence procedure is simple to implement and doesn't require storing any more information than is present in the weights and the threshold. The Gaussian classifier can be made adaptive [24], but extra information must be stored and the computations required are more complex.

Neither the perceptron convergence procedure nor the Gaussian classifier is appropriate when classes cannot be separated by a hyperplane. Two such situations are presented in the upper section of Fig. 14. The smooth closed contours labeled A and B in this figure are the input distributions for the two classes when there are two continuous valued inputs to the different nets. The shaded areas are the decision regions created by a single-layer perceptron and other feed-forward nets. Distributions for the two classes for the exclusive OR problem are disjoint and cannot be separated by a single straight line. This problem was used to illustrate the weakness of the perceptron by Minsky and Papert [32]. If the lower left B cluster is taken to be at the origin of this two dimensional space then the output of the classifier must be "high" only if one but not both of the inputs is "high". One possible decision region for class A which a perceptron might create is illustrated by the shaded region in the first row of Fig. 14. Input distributions for the second problem shown in this figure are meshed and also can not be separated by a single straight line. Situations similar to these may occur when parameters such as formant frequencies are used for speech recognition.

MULTI-LAYER PERCEPTRON

Multi-layer perceptrons are feed-forward nets with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the input and output nodes. A three-layer perceptron with two layers of hidden units is shown in Fig. 15. Multi-layer perceptrons overcome many of the limitations of single-layer perceptrons, but were generally not used in the past because effective training algorithms were not available. This has recently changed with the development of new training algorithms [40]. Although it cannot be proven that these algorithms converge as with single layer perceptrons, they have been shown to be successful for many problems of interest [40].

The capabilities of multi-layer perceptrons stem from the nonlinearities used within nodes. If nodes were linear elements, then a single-layer net with appropriately chosen weights could exactly duplicate those calculations performed by any multi-layer net. The capabilities of perceptrons with one, two, and three layers that use hard-limiting nonlinearities are illustrated in Fig. 14. The second column in this figure indicates the types of decision regions that can be formed with different nets. The next two columns

present examples of decision regions which could be formed for the exclusive OR problem and a problem with meshed regions. The rightmost column gives examples of the most general decision regions that can be formed.

As noted above, a single-layer perceptron forms half-plane decision regions. A two-layer perceptron can form any, possibly unbounded, convex region in the space spanned by the inputs. Such regions include convex polygons sometimes called convex hulls, and the unbounded convex regions shown in the middle row of Fig. 14. Here the term convex means that any line joining points on the border of a region goes only through points within that region. Convex regions are formed from intersections of the half-plane regions formed by each node in the first layer of the multi-layer perceptron. Each node in the first layer behaves like a single-layer perceptron and has a "high" output only for points on one side of the hyperplane formed by its weights and offset. If weights to an output node from N_1 first-layer nodes are all 1.0, and the threshold in the output node is $N_1 - \varepsilon$ where $0 < \varepsilon < 1$, then the output node will be "high" only if the outputs of all first-layer nodes are "high". This corresponds to performing a logical AND operation in the output node and results in a final decision region that is the intersection of all the half-plane regions formed in the first layer. Intersections of such half planes form convex regions as described above. These convex regions have at the most as many sides as there are nodes in the first layer.

This analysis provides some insight into the problem of selecting the number of nodes to use in a two-layer perceptron. The number of nodes must be large enough to form a decision region that is as complex as is required by a given problem. It must not, however, be so large that the many weights required can not be reliably estimated from the available training data. For example, two nodes are sufficient to solve the exclusive OR problem as shown in the second row of Fig. 14. No number of nodes, however, can separate the meshed class regions in Fig. 14 with a two-layer perceptron.

A three-layer perceptron can form arbitrarily complex decision regions and can separate the meshed classes as shown in the bottom of Fig. 14. It can form regions as complex as those formed using mixture distributions and nearest-neighbor classifiers [7]. This can be proven by construction. The proof depends on partitioning the desired decision region into small hypercubes (squares when there are two inputs). Each hypercube requires $2N$ nodes in the first layer (four nodes when there are two inputs), one for each side of the hypercube, and one node in the second layer that takes the logical AND of the outputs from the first-layer nodes. The outputs of second-layer nodes will be "high" only for inputs within each hypercube. Hypercubes are assigned to the proper decision regions by connecting the output of each second-layer node only to the output node corresponding to the decision region that node's hypercube is in and performing a logical OR operation in each output node. A logical OR operation will be performed if these connection weights from the second hidden layer to the output layer are one and thresholds in the output nodes are 0.5. This construction procedure can be generalized to use arbitrarily shaped convex regions instead of small hypercubes and is capable of generating the disconnected and non-convex regions shown at the bottom of Fig. 14.

The above analysis demonstrates that no more than three layers are required in perceptron-like feed-forward nets because a three-layer net can generate arbitrarily complex decision regions. It also provides some insight into the problem of selecting the number of nodes to use in three-layer perceptrons. The number of nodes in the second layer must be greater than one when decision regions are disconnected or meshed and cannot be formed from one convex area. The number of second layer nodes required in the worst case is equal to the number of disconnected regions in input distributions. The number of nodes in the first layer must typically be sufficient to provide three or more edges for each convex area generated by every second-layer node. There should thus typically be more than three times as many nodes in the second as in the first layer.

The above discussion centered primarily on multi-layer perceptrons with one output when hard limiting nonlinearities are used. Similar behavior is exhibited by multi-layer perceptrons with multiple output nodes when sigmoidal nonlinearities are used and the decision rule is to select the class corresponding to the output node with the largest output. The behavior of these nets is more complex because decision regions are typically bounded by smooth curves instead of by straight line segments and analysis is thus more difficult. These nets, however, can be trained with the new back-propagation training algorithm [40].

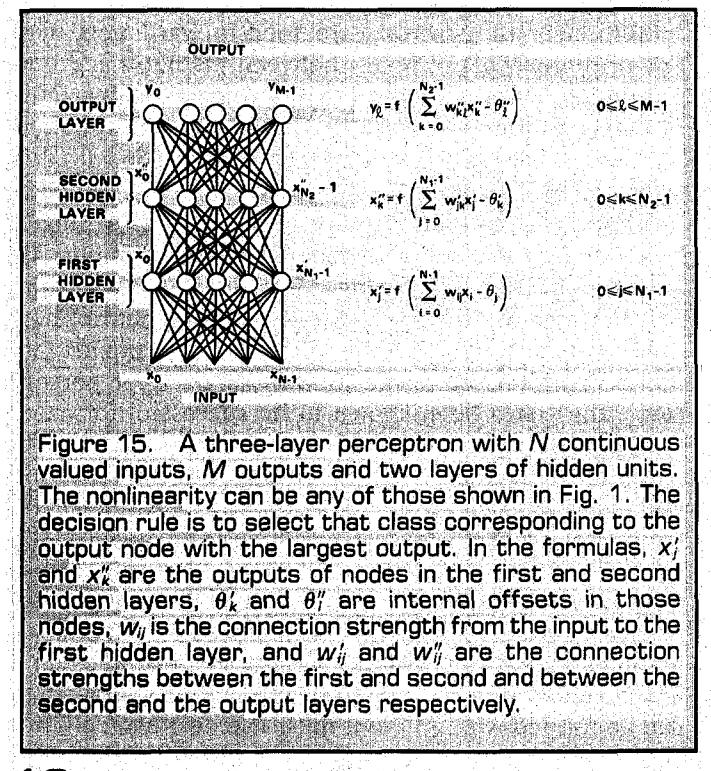

Figure 15. A three-layer perceptron with N continuous valued inputs, M outputs and two layers of hidden units. The nonlinearity can be any of those shown in Fig. 1. The decision rule is to select that class corresponding to the output node with the largest output. In the formulas, x'_j and x''_k are the outputs of nodes in the first and second hidden layers, θ'_k and θ''_ℓ are internal offsets in those nodes, w_{ij} is the connection strength from the input to the first hidden layer, and w'_{ij} and w''_{ij} are the connection strengths between the first and second and between the second and the output layers respectively.

The back-propagation algorithm described in Box 6 is a generalization of the LMS algorithm. It uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and the actual net outputs. The desired output of all nodes is typically "low" (0 or <0.1) unless that node corresponds to the class the current input is from in which case it is "high" (1.0 or >0.9). The net is trained by initially selecting small random weights and internal thresholds and then presenting all training data repeatedly. Weights are adjusted after every trial using side information specifying the correct class until weights converge and the cost function is reduced to an acceptable value. An essential component of the algorithm is the iterative method described in Box 6 that propagates error terms required to adapt weights back from nodes in the output layer to nodes in lower layers.

An example of the behavior of the back propagation algorithm is presented in Fig. 16. This figure shows decision regions formed by a two-layer perceptron with two inputs, eight nodes in the hidden layer, and two output nodes corresponding to two classes. Sigmoid nonlinearities were used as in Box 6, the gain term η was 0.3, the momentum term α was 0.7, random samples from classes A and B were presented on alternate trials, and the desired outputs were either 1 or 0. Samples from class A were distributed uniformly over a circle of radius 1 centered at the origin. Samples from class B were distributed uniformly outside this circle up to a radius of 5. The initial decision region is a slightly curved hyperplane. This gradually changes to a circular region that encloses the circular distribution of class A after 200 trails (100 samples from each class). This decision region is near that optimal region that would be produced by a Maximum Likelihood classifier.

The back propagation algorithm has been tested with a number of deterministic problems such as the exclusive OR problem [40], on problems related to speech synthesis

Box 6. The Back-Propagation Training Algorithm

The back-propagation training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multilayer feed-forward perceptron and the desired output. It requires continuous differentiable non-linearities. The following assumes a sigmoid logistic nonlinearity is used where the function $f(\alpha)$ in Fig. 1 is

$$f(\alpha) = \frac{1}{1 + e^{-(\alpha-\theta)}}$$

Step 1. Initialize Weights and Offsets

Set all weights and node offsets to small random values.

Step 2. Present Input and Desired Outputs

Present a continuous valued input vector $x_0, x_1, \ldots x_{N-1}$ and specify the desired outputs $d_0, d_1, \ldots d_{M-1}$. If the net is used as a classifier then all desired outputs are typically set to zero except for that corresponding to the class the input is from. That desired output is 1. The input could be new on each trial or samples from a training set could be presented cyclically until weights stabilize.

Step 3. Calculate Actual Outputs

Use the sigmoid nonlinearity from above and formulas as in Fig. 15 to calculate outputs $y_0, y_1, \ldots y_{M-1}$.

Step 4. Adapt Weights

Use a recursive algorithm starting at the output nodes and working back to the first hidden layer. Adjust weights by

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x'_i$$

In this equation $w_{ij}(t)$ is the weight from hidden node i or from an input to node j at time t, x'_i is either the output of node i or is an input, η is a gain term, and δ_j is an error term for node j. If node j is an output node, then

$$\delta_j = y_j(1 - y_j)(d_j - y_j),$$

where d_j is the desired output of node j and y_j is the actual output.

If node j is an internal hidden node, then

$$\delta_j = x_j(1 - x_j)\sum_k \delta_k w_{jk},$$

where k is over all nodes in the layers above node j. Internal node thresholds are adapted in a similar manner by assuming they are connection weights on links from auxiliary constant-valued inputs. Convergence is sometimes faster if a momentum term is added and weight changes are smoothed by

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x'_i + \alpha(w_{ij}(t) - w_{ij}(t-1)),$$

where $0 < \alpha < 1$.

Step 5. Repeat by Going to Step 2

and recognition [43, 37, 8] and on problems related to visual pattern recognition [40]. It has been found to perform well in most cases and to find good solutions to the problems posed. A demonstration of the power of this algorithm was provided by Sejnowski [43]. He trained a two-layer perceptron with 120 hidden units and more than 20,000 weights to form letter to phoneme transcription rules. The input to this net was a binary code indicating those letters in a sliding window seven letters long that was moved over a written transcription of spoken text. The desired output was a binary code indicating the phonemic transcription of the letter at the center of the window. After 50 times through a dialog containing 1024 words, the transcription error rate was only 5%. This increased to 22% for a continuation of that dialog that was not used during training.

The generally good performance found for the back propagation algorithm is somewhat surprising considering that it is a gradient search technique that may find a local minimum in the LMS cost function instead of the desired global minimum. Suggestions to improve performance and reduce the occurrence of local minima include allowing extra hidden units, lowering the gain term used to adapt weights, and making many training runs starting with different sets of random weights. When used with classification problems, the number of nodes could be set using considerations described above. The problem of local minima in this case corresponds to clustering two or more disjoint class regions into one. This can be minimized by using multiple starts with different random weights and a low gain to adapt weights. One difficulty noted with the backward-propagation algorithm is that in many cases the number of presentations of training data required for convergence has been large (more than 100 passes through all the training data). Although a number of more complex adaptation algorithms have been proposed to speed convergence [35] it seems unlikely that the complex decision regions formed by multi-layer perceptrons can be generated in few trials when class regions are disconnected.

An interesting theorem that sheds some light on the capabilities of multi-layer perceptrons was proven by Kolmogorov and is described in [26]. This theorem states that any continuous function of N variables can be computed using only linear summations and nonlinear but continuously increasing functions of only one variable. It effectively states that a three layer perceptron with $N(2N + 1)$ nodes using continuously increasing nonlinearities can compute any continuous function of N variables. A three-layer perceptron could thus be used to create any continuous likelihood function required in a classifier. Unfortunately, the theorem does not indicate how weights or nonlinearities in the net should be selected or how sensitive the output function is to variations in the weights and internal functions.

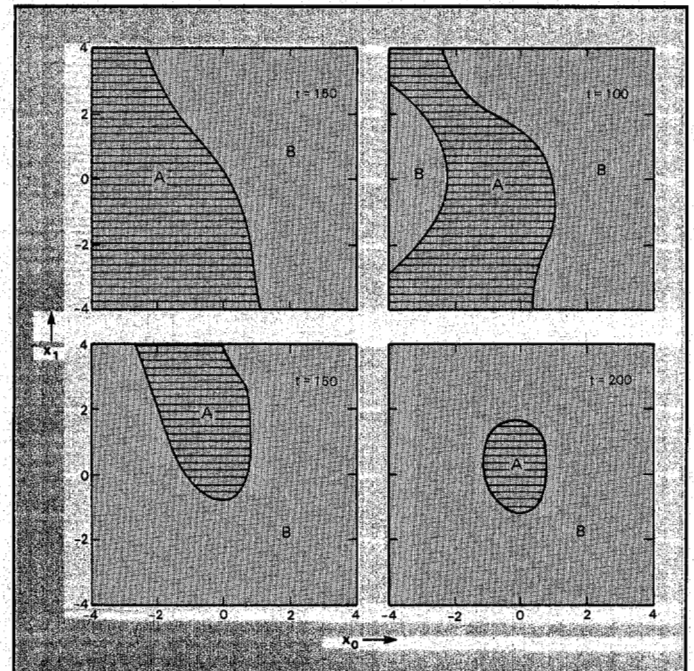

Figure 16. Decision regions after 50, 100, 150 and 200 trials generated by a two layer perceptron using the back-propagation training algorithm. Inputs from classes A and B were presented on alternate trials. Samples from class A were distributed uniformly over a circle of radius 1 centered at the origin. Samples from class B were distributed uniformly outside the circle. The shaded area denotes the decision region for class A.

KOHONEN'S SELF ORGANIZING FEATURE MAPS

One important organizing principle of sensory pathways in the brain is that the placement of neurons is orderly and often reflects some physical characteristic of the external stimulus being sensed [21]. For example, at each level of the auditory pathway, nerve cells and fibers are arranged anatomically in relation to the frequency which elicits the greatest response in each neuron. This tono-

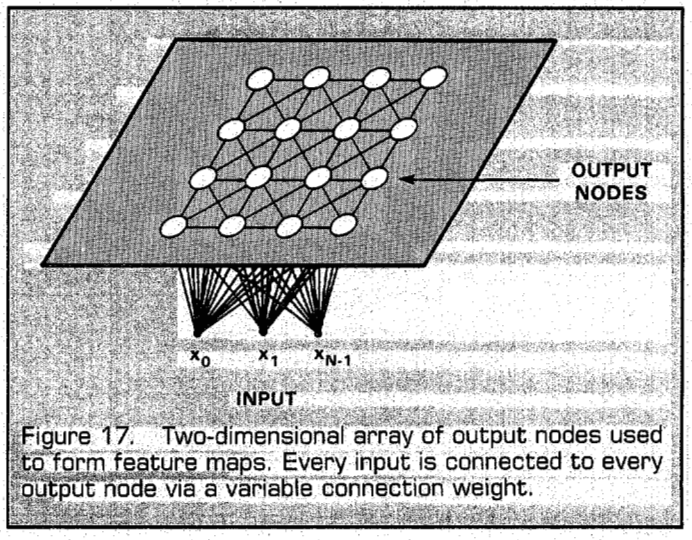

Figure 17. Two-dimensional array of output nodes used to form feature maps. Every input is connected to every output node via a variable connection weight.

topic organization in the auditory pathway extends up to the auditory cortex [33, 21]. Although much of the low-level organization is genetically pre-determined, it is likely that some of the organization at higher levels is created during learning by algorithms which promote self-organization. Kohonen [22] presents one such algorithm which produces what he calls self-organizing feature maps similar to those that occur in the brain.

Kohonen's algorithm creates a vector quantizer by adjusting weights from common input nodes to M output nodes arranged in a two dimensional grid as shown in Fig. 17. Output nodes are extensively interconnected with many local connections. Continuous-valued input vectors are presented sequentially in time without specifying the desired output. After enough input vectors have been presented, weights will specify cluster or vector centers that sample the input space such that the point density function of the vector centers tends to approximate the probability density function of the input vectors [22]. In addition, the weights will be organized such that topologically close nodes are sensitive to inputs that are physically similar. Output nodes will thus be ordered in a natural manner. This may be important in complex systems with many layers of processing because it can reduce lengths of inter-layer connections.

The algorithm that forms feature maps requires a neighborhood to be defined around each node as shown in Fig. 18. This neighborhood slowly decreases in size with time as shown. Kohonen's algorithm is described in Box 7. Weights between input and output nodes are initially set to small random values and an input is presented. The distance between the input and all nodes is computed as shown. If the weight vectors are normalized to have constant length (the sum of the squared weights from all inputs to each output are identical) then the node with the minimum Euclidean distance can be found by using the net of Fig. 17 to form the dot product of the input and the weights. The selection required in step 4 then turns into a problem of finding the node with a maximum value. This node can be selected using extensive lateral inhibition as in the MAXNET in the top of Fig. 6. Once this node is selected, weights to it and to other nodes in its neighborhood are modified to make these nodes more responsive to the current input. This process is repeated for further inputs. Weights eventually converge and are fixed after the gain term in step 5 is reduced to zero.

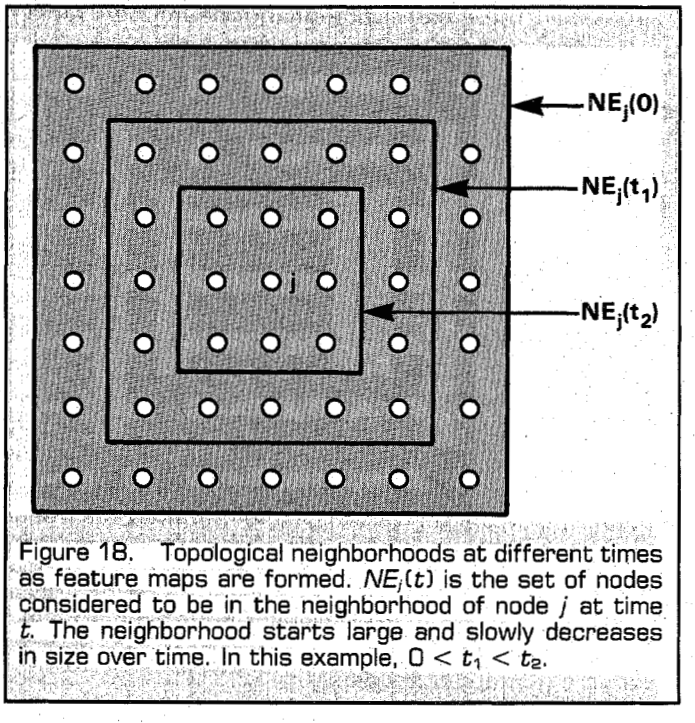

Figure 18. Topological neighborhoods at different times as feature maps are formed. $NE_j(t)$ is the set of nodes considered to be in the neighborhood of node j at time t. The neighborhood starts large and slowly decreases in size over time. In this example, $0 < t_1 < t_2$.

Box 7. An Algorithm to Produce Self-Organizing Feature Maps

Step 1. Initialize Weights

Initialize weights from N inputs to the M output nodes shown in Fig. 17 to small random values. Set the initial radius of the neighborhood shown in Fig. 18.

Step 2. Present New Input

Step 3. Compute Distance to All Nodes

Compute distances d_j between the input and each output node j using

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2$$

where $x_i(t)$ is the input to node i at time t and $w_{ij}(t)$ is the weight from input node i to output node j at time t.

Step 4. Select Output Node with Minimum Distance

Select node j^* as that output node with minimum d_j.

Step 5. Update Weights to Node j^* and Neighbors

Weights are updated for node j^* and all nodes in the neighborhood defined by $NE_{j^*}(t)$ as shown in Fig. 18. New weights are

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$

For $j \in NE_{j^*}(t)$ $\quad 0 \le i \le N-1$

The term $\eta(t)$ is a gain term $(0 < \eta(t) < 1)$ that decreases in time.

Step 6. Repeat by Going to Step 2

An example of the behavior of this algorithm is presented in Fig. 19. The weights for 100 output nodes are plotted in these six subplots when there are two random independent inputs uniformly distributed over the region enclosed by the boxed areas. Line intersections in these plots specify weights for one output node. Weights from

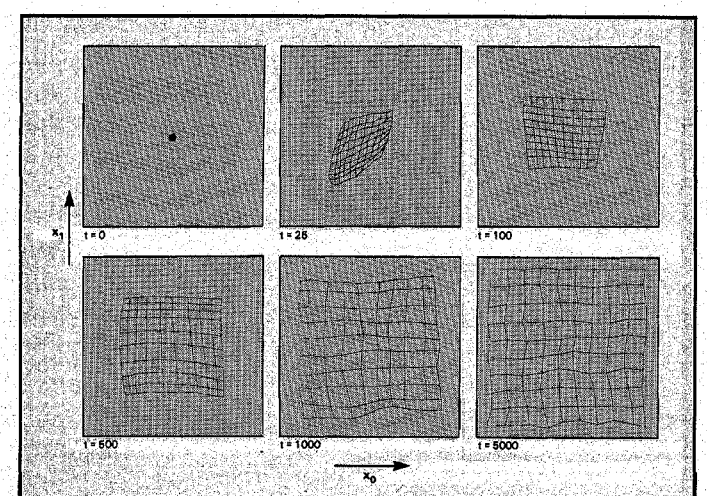

Figure 19. Weights to 100 output nodes from two input nodes as a feature map is being formed. The horizontal axis represents the value of the weight from input x_0 and the vertical axis represents the value of the weight from input x_1. Line intersections specify the two weights for each node. Lines connect weights for nodes that are nearest neighbors. An orderly grid indicates that topologically close nodes code inputs that are physically similar. Inputs were random, independent, and uniformly distributed over the area shown.

input x_0 are specified by the position along the horizontal axis and weights from input x_1 are specified by the position along the vertical axis. Lines connect weight values for nodes that are topological nearest neighbors. Weights start at time zero clustered at the center of the plot. Weights then gradually expand in an orderly way until their point density approximates the uniform distribution of the input samples. In this example, the gain term in step 5 of Box 7 was a Gaussian function of the distance to the node selected in step 4 with a width that decreased in time.

Kohonen [22] presents many other examples and proofs related to this algorithm. He also demonstrates how the algorithm can be used in a speech recognizer as a vector quantizer [23]. Unlike the Carpenter/Grossberg classifier, this algorithm can perform relatively well in noise because the number of classes is fixed, weights adapt slowly, and adaptation stops after training. This algorithm is thus a viable sequential vector quantizer when the number of clusters desired can be specified before use and the amount of training data is large relative to the number of clusters desired. It is similar to the *K*-means clustering algorithm in this respect. Results, however, may depend on the presentation order of input data for small amounts of training data.

INTRODUCTORY REFERENCES TO NEURAL NET LITERATURE

More detailed information concerning the six algorithms described above and other neural net algorithms can be found in [3, 7, 15, 18, 19, 20, 22, 25, 32, 39, 40]. Descriptions of many other algorithms including the Boltzmann machine and background historical information can be found in a recent book on parallel distributed processing edited by Rumelhart and McClelland [41]. Feldman [9] presents a good introduction to the connectionist philosophy that complements this book. Papers describing recent research efforts are available in the proceedings of the 1986 Conference on Neural Networks for Computing held in Snowbird, Utah [6]. Descriptions of how the Hopfield net can be used to solve a number of different optimization problems including the traveling salesman problem are presented in [20, 45]. A discussion of how content-addressable memories can be implemented using optical techniques is available in [1] and an introduction to the field of neurobiology is available in [21] and other basic texts.

In addition to the above papers and books, there are a number of neural net conferences being held in 1987. These include the "1987 Snowbird Meeting on Neural Networks for Computing" in Snowbird, Utah, April 1–5, the "IEEE First Annual International Conference on Neural Networks" in San Diego, California, June 21–24, and the "IEEE Conference on Neural Information Processing Systems — Natural and Synthetic" in Boulder, Colorado, November 8–12. This last conference is cosponsored by the IEEE Acoustics, Speech, and Signal Processing Society.

CONCLUDING REMARKS

The above review provides an introduction to an interesting field that is immature and rapidly changing. The six nets described are common components in many more complex systems that are under development. Although there have been no practical applications of neural nets yet, preliminary results such as those of Sejnowski [43] have demonstrated the potential of the newer learning algorithms. The greatest potential of neural nets remains in the high-speed processing that could be provided through massively parallel VLSI implementations. Several groups are currently exploring different VLSI implementation strategies [31, 13, 42]. Demonstrations that existing algorithms for speech and image recognition can be performed using neural nets support the potential applicability of any neural-net VLSI hardware that is developed.

The current research effort in neural nets has attracted researchers trained in engineering, physics, mathematics, neuroscience, biology, computer sciences and psychology. Current research is aimed at analyzing learning and self-organization algorithms used in multi-layer nets, at developing design principles and techniques to solve dynamic range and sensitivity problems which become important for large analog systems, at building complete systems for image and speech and recognition and obtaining experience with these systems, and at determining which current algorithms can be implemented using neuron-like components. Advances in these areas and in VLSI implementation techniques could lead to practical real-time neural-net systems.

ACKNOWLEDGMENTS

I have constantly benefited from discussions with Ben Gold and Joe Tierney. I would also like to thank Don Johnson for his encouragement, Bill Huang for his simulation studies, and Carolyn for her patience.

REFERENCES

[1] Y. S. Abu-Mostafa and D. Pslatis, "Optical Neural Computers," *Scientific American,* 256, 88–95, March 1987.

[2] E. B. Baum, J. Moody, and F. Wilczek, "Internal Representations for Associative Memory," NSF-ITP-86-138 Institute for Theoretical Physics, University of California, Santa Barbara, California, 1986.

[3] G. A. Carpenter, and S. Grossberg, "Neural Dynamics of Category Learning and Recognition: Attention, Memory Consolidation, and Amnesia," in J. Davis, R. Newburgh, and E. Wegman (Eds.) *Brain Structure, Learning, and Memory,* AAAS Symposium Series, 1986.

[4] M. A. Cohen, and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Trans. Syst. Man Cybern.* SMC-13, 815–826, 1983.

[5] B. Delgutte, "Speech Coding in the Auditory Nerve: II. Processing Schemes for Vowel-Like Sounds," *J. Acoust. Soc. Am.* 75, 879–886, 1984.

[6] J. S. Denker, *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.

[7] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* John Wiley & Sons, New York (1973).

[8] J. L. Elman and D. Zipser, "Learning the Hidden Structure of Speech," Institute for Cognitive Science, University of California at San Diego, *ICS Report 8701,* Feb. 1987.

[9] J. A. Feldman and D. H. Ballard, "Connectionist Models and Their Properties," *Cognitive Science, Vol. 6,* 205–254, 1982.

[10] R. G. Gallager, *Information Theory and Reliable Communication,* John Wiley & Sons, New York (1968).

[11] O. Ghitza, "Robustness Against Noise: The Role of Timing-Synchrony Measurement," in *Proceedings International Conference on Acoustics Speech and Signal Processing,* ICASSP-87, Dallas, Texas, April 1987.

[12] B. Gold, "Hopfield Model Applied to Vowel and Consonant Discrimination," *MIT Lincoln Laboratory Technical Report, TR-747, AD-A169742,* June 1986.

[13] H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant, and D. Schwartz, "VLSI Implementation of a Neural Network Memory With Several Hundreds of Neurons," in J. S. Denker (Ed.) *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.

[14] P. M. Grant and J. P. Sage, "A Comparison of Neural Network and Matched Filter Processing for Detecting Lines in Images," in J. S. Denker (Ed.) *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.

[15] S. Grossberg, *The Adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythm,* and *The Adaptive Brain II: Vision, Speech, Language, and Motor Control,* Elsevier/North-Holland, Amsterdam (1986).

[16] J. A. Hartigan, *Clustering Algorithms,* John Wiley & Sons, New York (1975).

[17] D. O. Hebb, *The Organization of Behavior,* John Wiley & Sons, New York (1949).

[18] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci.* USA, Vol. 79, 2554–2558, April 1982.

[19] J. J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci.* USA, Vol. 81, 3088–3092, May 1984.

[20] J. J. Hopfield, and D. W. Tank, "Computing with Neural Circuits: A Model," *Science,* Vol. 233, 625–633, August 1986.

[21] E. R. Kandel and J. H. Schwartz, *Principles of Neural Science,* Elsevier, New York (1985).

[22] T. Kohonen, *Self-Organization and Associative Memory,* Springer-Verlag, Berlin (1984).

[23] T. Kohonen, K. Masisara and T. Saramaki, "Phonotopic Maps—Insightful Representation of Phonological Features for Speech Representation," *Proceedings IEEE 7th Inter. Conf. on Pattern Recognition,* Montreal, Canada, 1984.

[24] F. L. Lewis, *Optimal Estimation,* John Wiley & Sons, New York (1986).

[25] R. P. Lippmann, B. Gold, and M. L. Malpass, "A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification," *MIT Lincoln Laboratory Technical Report, TR-769,* to be published.

[26] G. G. Lorentz, "The 13th Problem of Hilbert," in F. E. Browder (Ed.), *Mathematical Developments Arising from Hilbert Problems,* American Mathematical Society, Providence, R.I. (1976).

[27] R. F. Lyon and E. P. Loeb, "Isolated Digit Recognition Experiments with a Cochlear Model," in *Proceedings International Conference on Acoustics Speech and Signal Processing, ICASSP-87,* Dallas, Texas, April 1987.

[28] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," *IEEE Proceedings, 73,* 1551–1588, Nov. 1985.

[29] T. Martin, *Acoustic Recognition of a Limited Vocabulary in Continuous Speech,* Ph.D. Thesis, Dept. Electrical Engineering Univ. Pennsylvania, 1970.

[30] W. S. McCulloch, and W. Pitts, "A Logical Calculus of the Ideas Imminent in Nervous Activity," *Bulletin of Mathematical Biophysics,* 5, 115–133, 1943.

[31] C. A. Mead, *Analog VLSI and Neural Systems,* Course Notes, Computer Science Dept., California Institute of Technology, 1986.

[32] M. Minsky, and S. Papert, *Perceptrons: An Intro-*

duction to Computational Geometry, MIT Press (1969).
[33] A. R. Moller, *Auditory Physiology,* Academic Press, New York (1983).
[34] P. Mueller, and J. Lazzaro, "A Machine for Neural Computation of Acoustical Patterns with Application to Real-Time Speech Recognition," in J. S. Denker (Ed.) *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.
[35] D. B. Parker, "A Comparison of Algorithms for Neuron-Like Cells," in J. S. Denker (Ed.) *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.
[36] T. Parsons, *Voice and Speech Processing,* McGraw-Hill, New York (1986).
[37] S. M. Peeling, R. K. Moore, and M. J. Tomlinson, "The Multi-Layer Perceptron as a Tool for Speech Pattern Processing Research," in *Proc. IoA Autumn Conf. on Speech and Hearing,* 1986.
[38] T. E. Posch, "Models of the Generation and Processing of Signals by Nerve Cells: A Categorically Indexed Abridged Bibliography," *USCEE Report 290,* August 1968.
[39] R. Rosenblatt, *Principles of Neurodynamics,* New York, Spartan Books (1959).
[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation" in D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations.* MIT Press (1986).
[41] D. E. Rumelhart, and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* MIT Press (1986).
[42] J. P. Sage, K. Thompson, and R. S. Withers, "An Artificial Neural Network Integrated Circuit Based on MNOS/CD Principles," in J. S. Denker (Ed.) *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP,* 1986.
[43] T. Sejnowski and C. R. Rosenberg, "NETtalk: A Parallel Network That Learns to Read Aloud," *Johns Hopkins Univ. Technical Report JHU/EECS-86/01,* 1986.
[44] S. Seneff, "A Computational Model for the Peripheral Auditory System: Application to Speech Recognition Research," in *Proceedings International Conference on Acoustics Speech and Signal Processing, ICASSP-86,* 4, 37.8.1-37.8.4, 1986.
[45] D. W. Tank and J. J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. Circuits Systems CAS-33,* 533–541, 1986.
[46] D. J. Wallace, "Memory and Learning in a Class of Neural Models," in B. Bunk and K. H. Mutter (Eds.) *Proceedings of the Workshop on Lattice Gauge Theory, Wuppertal, 1985,* Plenum (1986).
[47] B. Widrow, and M. E. Hoff, "Adaptive Switching Circuits," *1960 IRE WESCON Conv. Record, Part 4,* 96–104, August 1960.
[48] B. Widrow and S. D. Stearns, *Adaptive Signal Processing,* Prentice-Hall, New Jersey (1985).

Richard P. Lippmann (M'85) was born in Mineola, NY, in 1948. He received the B.S. degree in electrical engineering from the Polytechnic Inst. of Brooklyn in 1970 and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, in 1973 and 1978 respectively. His S.M. thesis dealt with the psychoacoustics of intensity perception and his Ph.D. thesis with signal processing for the hearing impaired.

From 1978 to 1981 he was the Director of Communication Engineering Laboratory at the Boys Town Institute for Communication Disorders in Children, in Omaha, NE. He worked on speech recognition, speech training aids for deaf children, sound alerting aids for the deaf, and signal processing for hearing aids. In 1981 he joined the MIT Lincoln Laboratory in Lexington, MA. He has worked on speech recognition, on speech I/O systems, and on routing and system control of circuit-switched networks. His current interests include speech recognition, neural net algorithms, statistics, and human physiology, memory, and learning.