Naval Ocean Systems Center
San Diego, CA 92152-5000

(4)

Technical Document 1335
August 1988

Adaptive Gaussian Pattern Classification

C. E. Priebe
D. J. Marchette

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

ADMINISTRATIVE INFORMATION

Information in this report originally was prepared by members of the Architecture and Applied Research Branch (NOSC Code 421) to be presented as a professional paper in the IEEE Transactions on Pattern Analysis and Machine Intelligence.

Released by
V. J. Monteleon, Head
Architecture and Applied Research Branch

Under authority of
J. A. Salzmann, Jr., Head
Information Systems Division

LO

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NOSC Technical Document 1335	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Ocean Systems Center — 6b. OFFICE SYMBOL (if applicable) Code 421	7a. NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State and ZIP Code) San Diego, CA 92152-5000	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Chief of Naval Operations — 8b. OFFICE SYMBOL (if applicable) CNO-OP-009V	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) Department of Navy Washington, DC 20350	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 30000N; PROJECT NO. RDN; TASK NO. CD59; AGENCY ACCESSION NO. ICCD5900

11. TITLE (include Security Classification)
ADAPTIVE GAUSSIAN PATTERN CLASSIFICATION

12. PERSONAL AUTHOR(S)
C. E. Priebe and D. J. Marchette

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM TO	August 1988	48

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
			Gaussian distributions density estimation pattern recognition

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A massively parallel architecture for pattern classification is described. The architecture is based on the field of density estimation. It makes use of a variant of the adaptive kernel estimator to approximate the distributions of the classes as a sum of Gaussian distributions. These Gaussians are learned using a moving mean, moving covariance learning scheme. A temporal ordering scheme is implemented using decay at the input level, allowing the network to learn to recognize sequences. The learning scheme requires a single pass through the data, giving the architecture the capability of real-time learning. The first part of the paper develops the adaptive kernel estimator. The paralle architecture is then described, and issues relevant to implementation are discussed. Finally, applications to robotic sensor fusion, isolated word recognition, and vision are described.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL C. E. Priebe		22b. TELEPHONE (include Area Code) (619) 553-4048	22c. OFFICE SYMBOL Code 421

DD FORM 1473, 84 JAN — 83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

Adaptive Gaussian Pattern Classification

C. E. Priebe
D. J. Marchette

Abstract -- A massively parallel architecture for pattern classification is described. The architecture is based on the field of density estimation. It makes use of a variant of the adaptive kernel estimator to approximate the distributions of the classes as a sum of Gaussian distributions. These Gaussians are learned using a moving mean, moving covariance learning scheme. A temporal ordering scheme is implemented using decay at the input level, allowing the network to learn to recognize sequences. The learning scheme requires a single pass through the data, giving the architecture the capability of real-time learning. The first part of the paper develops the adaptive kernel estimator. The parallel architecture is then described, and issues relevant to implementation are discussed. Finally, applications to robotic sensor fusion, isolated word recognition, and vision are described.

I. Introduction

This paper describes a pattern recognition technique which arises from the related area of probability density estimation. The technique is described within the framework of a distributed algorithm implemented as an Adaptive Network System (ANS) (also called "adaptive neural system" or "neural network"). This gives the potential for real-time processing, and the system is truly adaptive in the sense that "learning" need never stop. The paper is organized around the two concepts of density estimation and network architectures. Density estimation and several related methods are described in the context of a network architecture. Finally, applications to real world problems using architectures tuned to the particular problem are described.

What is density estimation?

The problem of estimating the probability density function of a distribution from a finite set of points sampled from the distribution is fundamental to many applications. Such an estimate gives a measure of the variation of the data, the number of modes in the data, and allows comparisons between different data sets which can be of value in classification tasks.

A-1

The probability density function f of a random variable X is a non-negative real valued function described by the relationship:

$$P(a<X<b) = \int_a^b f(x)\,dx \tag{1}$$

where P represents probability. Density estimation is the technique of estimating the probability density function of a distribution from a finite set of points. The most familiar technique is the histogram method. Essentially, the histogram method partitions the input space into intervals, and a count is kept of the number of points in each interval. The counts are normalized by the total number of points multiplied by the width of the intervals. This gives a histogram which in many cases is a good estimation for the density. An obvious problem with this approach is that the size of the interval is critical to the performance of the estimator (in fact this is a problem common to many estimators). To see this, consider the two extremes of an interval size, one which is so large that all points lie in a single interval, and in the other extreme an interval size so small that each point lies in a unique interval. The problem now is to choose the interval size in such a way that one arrives at a good estimate of the density. Much work has been done in this area [13]. This is the problem that we are addressing with our architecture, though it is based on the kernel estimator, which is an improvement over the histogram method, and uses Gaussians instead of rectangles to build up an estimate of the distribution.

What is classification?

Classification involves assigning a class name to each element of a set of data. In addition to the assignment of a class, it is desirable to give a probability or likelihood that the assignment is correct. In general, some measure of likelihood is given for all classes from which the data point could have come. This vector of likelihood measures for each of the classes will be referred to as the classification vector.

Examples of classification problems are found in many fields. A doctor would like to classify a patient as high risk or low risk. Stars are classified according to their emissions, size etc. Clouds are classified as cumulus, stratus, etc., and this information has consequences in the prediction of weather. Speech processing involves the classification of a signal as a particular utterance, or as coming from an individual speaker. All of these problems involve assigning a class name to a data point. The solutions to these problems involve studying a set of data for which truth is known, and constructing a model of the various classes which allows the assignment of a name to each data point input. One of the most natural ways to do this is to construct the probability density functions for the different classes, and return the classification vector for that input. This is the approach that will be explored in this paper.

Overview of network architectures

Network architectures, which fall under various labels such as "neural networks", "adaptive network systems", "adaptive neural systems", and many others, all have a common basic architecture (see, for example, [11]). This consists of a collection of independent processing elements, called "nodes" (also called "neurons" by some) and connections between nodes through which they communicate. Specifically, a node is a processing element which has any number of inputs and a single output (Figure 1). The node performs some operation on its inputs, and sends the results of this operation out. There is a so called "local principle" invoked which states that the operation performed by the node is dependent on its inputs and its current state alone. No global information is allowed, and the states of other nodes cannot be used except as they are transmitted as inputs to this node. The inputs travel along "weighted connections", which can be thought of as wires with a resistance associated with them. The output of a node gets broadcast to one or more other nodes along these connections, with each connection weighting this output. The input to a node, then, is the weighted outputs of other nodes.

Note that Figure 1 indicates that the output of the node is an arbitrary function of the input and weight vectors. Though this is a reasonable construction in general, the function F is usually more restrictive. In most architectures, F is a function of $\mathbf{I}\cdot\mathbf{W}$. In our architecture, F is a function of $\mathbf{I} - \mathbf{W}$. In addition to these vectors, F may depend on locally stored information, which is not indicated in the figure.

The basic architecture is then a directed graph of weighted connections, with processing occuring at each node in the graph (Figure 2). The network is divided into layers of nodes, with the nodes in a layer having similar properties. The connections conveying input into a node are referred to as the afferent connections of the node, and the connections projecting out from a node are referred to as the efferent connections of the node.

Associated with this graph is a learning rule which describes the manner in which the connection weights and any variables local to a node are changed as data flows through the network. Typically, a training set of data, a truth set of known data or subset of all available data, is used to teach the network a desired computation, and after training the network is tested on a separate set of the same type of data. Another technique of validation, referred to as cross-validation, is to teach the system on all but one datum, then test on this datum. This process

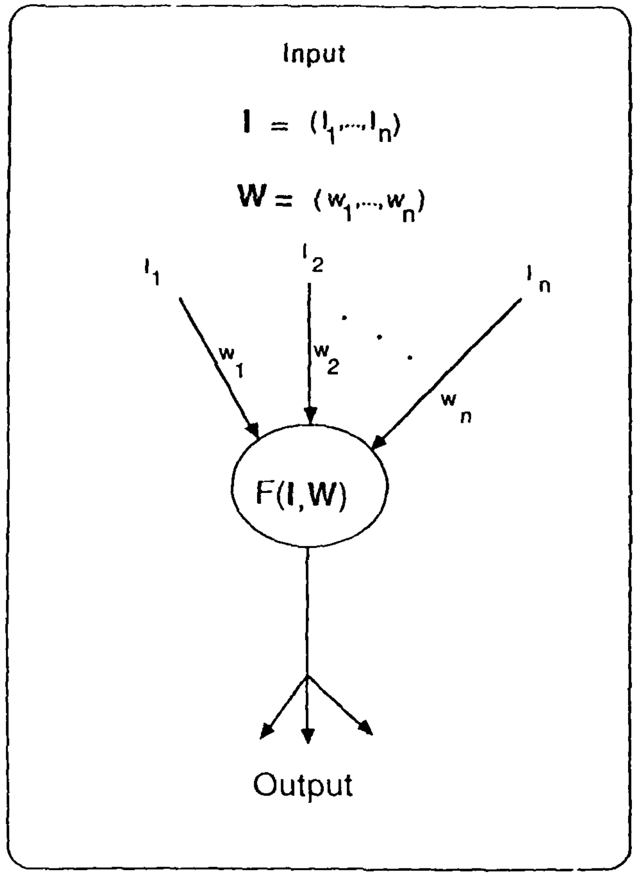
Input
$\mathbf{I} = (I_1,\ldots,I_n)$
$\mathbf{W} = (w_1,\ldots,w_n)$
I_1
I_2
I_n
w_1
w_2
w_n
F(I,W)
Output

Figure 1. Diagram of a single node. The output is a function of the input vector (the outputs of other nodes) and a weight vector. The output can also depend on variables local to the node, though this is not made explicit in the figure.

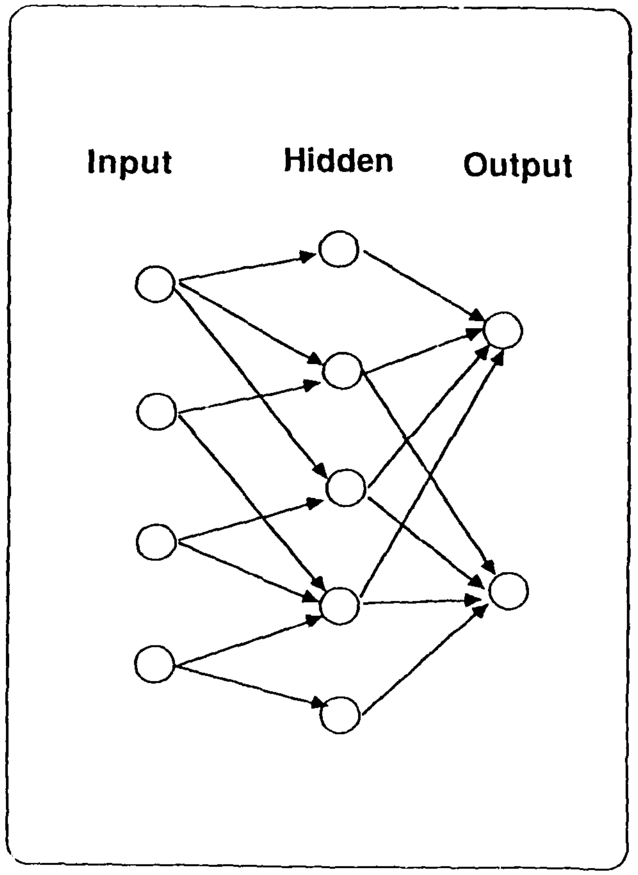

Figure 2. A feedforward network. Nodes are represented as circles and connections are represented as arrows.

is repeated until all the points have been tested. This technique is useful in the event that the available data is not large enough to allow for significant teaching and test sets.

II. Gaussian Classification

The Gaussian or normal distribution (2) is used extensively in density estimation. In parametric density estimation, the distribution is assumed to be a known one, in this case a Gaussian, and the parameters of the distribution are estimated from the data. A more general approach is to approximate the distribution as a sum of Gaussians. This allows the modelling of nearly arbitrary distributions in principle, though it can have problems in practice. Distributions for which some moments do not exist, and distributions with finite support cannot be represented as sums of Gaussians, but the approximation as a sum of Gaussians may be good enough for most purposes. The kernel estimator is the most commonly used method of this type.

The multivariate, or multi-dimensional, Gaussian density is given by:

$$G(\underline{\mathbf{x}}) = \frac{\exp\left(-0.5 \left[(\underline{\mathbf{x}} - \underline{\mu})^t \cdot \Sigma^{-1} \cdot (\underline{\mathbf{x}} - \underline{\mu}) \right] \right)}{(2\pi)^{(d/2)} * |\Sigma|^{(1/2)}} \qquad (2)$$

Here, $\underline{\mathbf{x}}$ is a real valued vector of dimension **d**, $\underline{\mu}$ is the mean of the Gaussian, and Σ is the covariance matrix. The expression in square brackets above is referred to as the Mahalanobis distance of $\underline{\mathbf{x}}$ from $\underline{\mu}$. Equation 2 is the multi-dimensional analog of the familiar normal distribution.

One of the applications of density estimation is classification. The classifier partitions the input space into regions corresponding to the different classes. For example, given a set of measurements such as blood pressure, heart rate, etc. a doctor would like to classify a heart patient as high risk or low risk. Given a historical database of previous patients, one could model the probability density for each of the two categories, then decide the class based on the relative values of the estimators on the current patient's data. The absolute values of the outputs, combined with the overall shapes of the distributions, gives a good estimate for the likelihood that the point is in fact from the chosen class. Figure 3 shows the distributions which may be arrived at from a sample of such data. Note that mistaken conclusions can arise from not observing the full distribution. For example, a point near the mean of the high risk patients can give a larger value for the low risk distribution, a potentially dangerous occurance in the event the point actually represents a high risk patient. This indicates that the overall shape of the distribution is an important factor in the classification decision. The doctor can weight this information with the penalty for a wrong assessment to produce a more reliable diagnosis.

The idea fundamental to the work described here is the one of using Gaussian distributions as the blocks to build up the distribution. The basic idea is to model the distribution as a sum of Gaussian distributions (Figure 4), in much the same way that the histogram method is a sum of rectangles. Thus, the estimation is made up of a series of Gaussian "bumps", and the problem is to find the appropriate Gaussians to model a given data set. It is clear that (given reasonable assumptions on the distribution as mentioned above) any distribution can be approximated by sums of Gaussians (Figure 4).

Gaussians as feature detectors

Another interpretation of the Gaussians in this scheme is as basic feature detectors. Here the goas is classification, rather than density estimation for its own sake. Each Gaussian (2) can be thought of as keying on a particular feature within the data, and the number of Gaussians used for the data determines the number of features used to identify the classes. In this interpretation, the means of the Gaussians become the features, and each Gaussian returns a distance measure between its feature and the input data item. It is instructive to think of the Gaussians in this manner, particularly as it gives them an autonomous flavor. Each Gaussian is conceived of as comparing the input with its stored feature, making its computation independently of the others. It is this independence that is critical to implementation as a network architecture and leads to the architecture that will be described.

An interesting interpretation of the Gaussian nodes is as a Voronoi, or nearest neighbor classifier [7]. Consider a set of points, each with a single Gaussian over them (Figure 5), where the level curves of the Gaussians are circles (in general the level curves are ellipses). If the Gaussians are identical except for their means, then for any set of level curves, the intersections define the boundaries of a Voronoi classifier; that is, the Gaussian nodes produce an optimal nearest neighbor classifier. If each point is considered to be a typical example of a particular feature within the data, then the Gaussian above the point can be considered a detector of that feature. It is in this sense that the Gaussian nodes in the architecture described below can be considered to be feature detectors.

There are two extremes in the use of Gaussians for classification. The assumption can be made that the classes all come from normal distributions, and a single Gaussian can be assigned to each class, with its mean and covariance matrix estimated from the data. In the other extreme, each point of the data can be assigned a Gaussian, and the Gaussians for each class are summed to

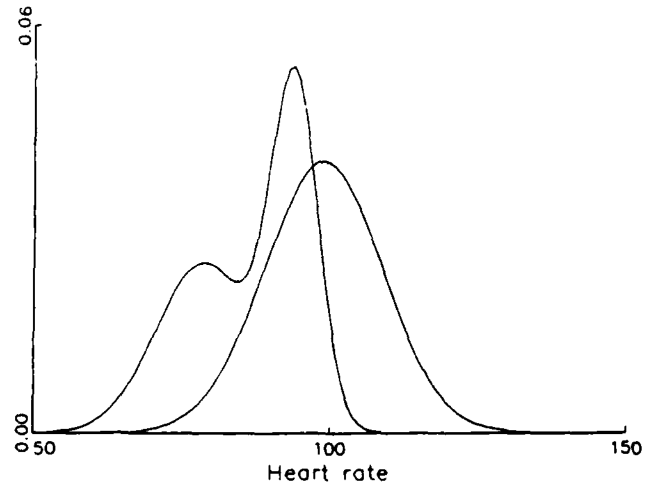

Figure 3. Densities of high and low risk heart patients from a fictional study. The unimodal distribution represents high risk patients and the bimodal represents low risk patients.

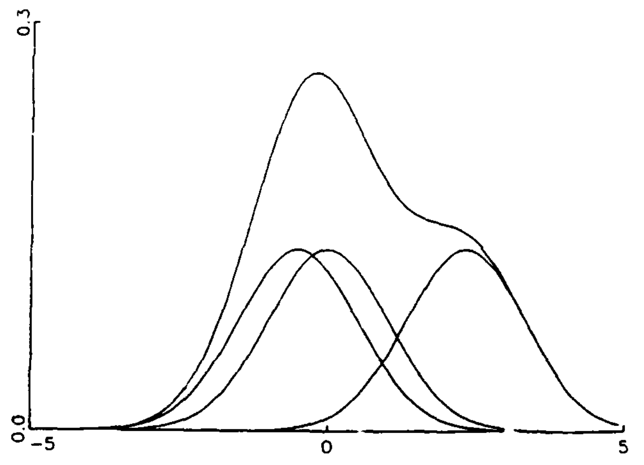

Figure 4. Representation of a bimodal distribution as the sum of Gaussian distributions.

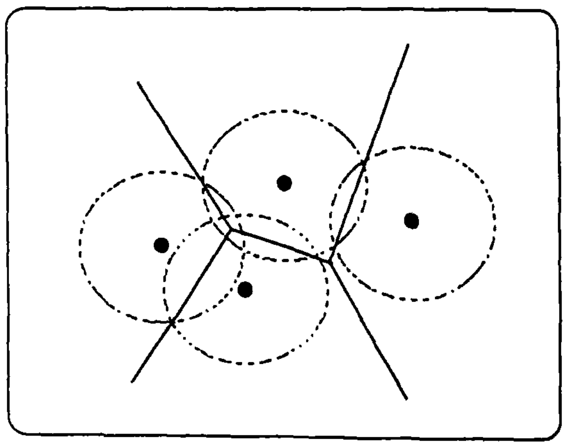

Figure 5. Voronoi classification using Gaussians. The circles represent level curves of the Gaussians centered over the solid dots. The line segments show the Voronoi classification.

produce the estimate of the density. In this case, the data point becomes the mean for the Gaussian and the covariance matrix is determined experimentally.

A more general case, subsuming both, is the architecture which will be described in the sequel. Many choices can be made to determine the number of Gaussians to be used in the approximation, and an adaptive method is described in which the number of Gaussians is determined by the distribution of the data. This produces a dynamic, rather than static, system which adapts itself to the data. The main distinction between static and dynamic systems is that a static system cannot adapt to new data, while a dynamic one is allowed to modify its representation after the initial teaching.

III. Single Gaussian distribution

Fitting a Gaussian to data

If *a priori* information about the distribution leads to the assumption that it is a Gaussian distribution, or if this assumption is not too restrictive, it is reasonable to model the data as a Gaussian distribution. This involves using a sample set to determine the multivariate mean and covariance matrix. There are several ways to do this, but the simplest is to proceed from the definitions. Let $\underline{x}$ be a random vector of m components, and let $\{\underline{x}_k\}$ be n vectors sampled from the distribution of $\underline{x}$. Then the sample mean is defined to be the vector $\underline{\mu}$ whose components are

$$\mu_i = \frac{1}{n} \sum_{k=1}^{n} x_{ki}, \qquad 1 \leq i \leq m \tag{3}$$

where x_{ki} is the i^th component of the k^th sample vector $\underline{x}_k$, and the sample covariance matrix is the matrix Σ whose components are

$$\Sigma_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (x_{ki} \ \ \mu_i)(x_{kj} - \mu_j), \quad 1 \leq i \leq m,\ 1 \leq j \leq m \tag{4}$$

This is the unbiased form of the covariance [12]. From these definitions, it is clear how an estimate of the distribution would be formed.

Adaptive Gaussian: How to adapt the Gaussians' parameters to new data

Now that the Gaussians' parameters have been determined, how can they be changed to adapt to new data? The answer is the key to the network solution proposed in Section V. First, consider the mean μ. From (3) we have, after n+1 points:

$$\mu_i(n+1) = \frac{1}{n+1} \sum_{k=1}^{n+1} x_{ki} \tag{5}$$

$$= \frac{1}{n+1} \sum_{k=1}^{n} x_{ki} + \frac{1}{n+1} x_{(n+1)i} \tag{6}$$

$$= \frac{n}{n+1} \mu_i(n) + \frac{1}{n+1} x_{(n+1)i} \tag{7}$$

$$= \mu_i(n) + \frac{1}{n+1} (x_{(n+1)i} - \mu_i(n)) \tag{8}$$

and so the change in μ, $\Delta\mu$, is defined by

$$\Delta\mu_i(n+1) = \frac{1}{n+1} (x_{(n+1)i} - \mu_i(n)) \tag{9}$$

Equation (9) gives the amount that the mean should be changed to include the new data point. A similar formula can be constructed from the sample covariance [1]. This formula hinges on the computation of the sums S_{ij}:

$$S_{ij}(n+1) = \sum (x_{ki} - \mu_i(n+1))(x_{kj} - \mu_j(n+1)) \tag{10}$$

The summations will always be from 1 to n+1 in what follows. With this definition, the covariance after n points is either S(n+1)/n-1 or S(n+1)/n, depending on whether the unbiased or biased estimator is desired [12]. The iterative formula for S is then derived from (10) as follows. Let

$$\alpha_{kl} = x_{kl} - \mu_l(n) \tag{11}$$

Then, replacing $\underline{\mu}$ in (10) with formula (8), we have

$$S_{ij}(n+1) = \sum[\alpha_{ki} - \frac{1}{n+1}\alpha_{(n+1)i}][\alpha_{kj} - \frac{1}{n+1}\alpha_{(n+1)j}] \tag{12}$$

$$= \sum\alpha_{ki}\,\alpha_{kj} + \frac{1}{n+1}[-\alpha_{(n+1)i}\sum\alpha_{kj} - \alpha_{(n+1)j}\sum\alpha_{ki} + \alpha_{(n+1)i}\,\alpha_{(n+1)i}] \tag{13}$$

$$= S_{ij}(n) + \alpha_{(n+1)i}\,\alpha_{(n+1)j} - \frac{1}{n+1}\alpha_{(n+1)i}\,\alpha_{(n+1)j} \tag{14}$$

since the sums in parentheses in (13) are equal to $\alpha_{(n+1)l}$ with l = i or j, as appropriate. This leads to the update formula:

$$S_{ij}(n+1) = S_{ij}(n) + \frac{n}{n+1}(x_{(n+1)i} - \mu_i(n))\,(x_{(n+1)j} - \mu_j(n+1)) \tag{15}$$

This formula gives $\underline{S}$ in a form which can be calculated independently of the update calculation of the mean, and hence can be carried out in parallel with the mean calculation. In the sequel the formula for unbiased covariance will be used:

$$\Sigma_{ij} = \frac{1}{n} S_{ij} \tag{16}$$

after n+1 points.

Formulas (9) and (15) give an iterative method for determining the mean and covariance for the case of an estimate consisting of a single Gaussian. In Section V these formulas will form the basis for the learning rule in a system which uses a sum of Gaussians for its representation.

IV. Kernel Estimators

One of the most popular density estimation techniques is the kernel estimator. For clarity, the system for a one-dimensional distribution will be described. This extends to multiple dimensions, though there are serious problems with density estimation in high dimensions.

The kernel estimator can be thought of as a generalization of the single Gaussian construction above. The idea is to approximate the distribution as a sum of Gaussians. This allows for approximation of arbitrary distributions. Just as the histogram method approximates the distribution as a sum of rectangles, the kernel estimator uses a sum of "bumps". These "bumps" can have width varying from near zero, in which case the sum is, roughly speaking, a sum of Dirac delta functions [13], to very large with respect to the range of the distribution, in which case all detail is obscured and the result is similar to a single Gaussian.

The kernel estimator is constructed by assigning to each point in the sample distribution a Gaussian. The estimation is then

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K(x-X_i/h) \tag{17}$$

where h is window width, or variance, X_i is the sample point, n is the number of samples, and K(x) is the kernel, in this case the Gaussian

$$K(x) = \frac{\exp(-x^2/2)}{(2\pi)^{1/2}} \tag{18}$$

In general, almost any distribution could be used for the kernel, but for the purposes of this paper K will always be a Gaussian.

Assuming that the number of samples is fixed, the only variable quantity in (17) is the window width h. Different values of h will give different estimators. The kernel estimator is widely used in practice, though it does suffer from the fact that a single window width is used for all points (see [13] for adaptive kernel techniques which allow varying the window width). The main problem with these techniques is that they require a large number of nodes in a network implementation, which is one of the motivations for the architecture suggested in Section V.

V. Network architecture

The actual computational architecture proposed is designed explicitly for massively parallel processing. The application areas in pattern recognition, specifically vision and audition, require enormous processing resources. Only by designing a system with a large number of independent units, and therefore a high degree of distributed processing, can we hope to approach a real-time solution to these problems.

The adaptive Gaussian architecture was developed from the principles of neural network theory, but has a number of fundamental differences. Nevertheless, the basic network features of information flow can be used to understand the dynamics of the system. The system is described as a collection of nodes, organized in layers, and connections between these nodes. Preliminary work can be found in [8]. The design proposed, as well as the most promising application areas, suggest a discrete-time system in which each node performs some function at each time step. This does not preclude asynchronous behavior, although the descriptions herein are for synchronous nodal processing.

The nodes in the network must be considered independent of one another for the purposes of nodal processing. Each node performs its computations at a given time step based on inputs from an explicit set of surrounding nodes. As such, this system can be thought of as a type of cellular automaton [2], albeit with a somewhat complex, and in fact dynamic, neighborhood (in a network such as this, the neighborhood of G_j is all nodes with efferent connections to G_j). In addition, if G_j uses its own value at time t to produce a value at time t+1, then G_j is considered a member of its own neighborhood.

The network is organized in layers of nodes, with the information flowing from the input layer, through the Gaussian layers (hidden layers), and finally yielding a network output at the output layer (Figure 6). First the individual nodes will be discussed in detail, and this understanding of the processing capabilities of each member of the network will then be used to understand the network organization from a macroscopic viewpoint.

Individual nodes

In a description of any highly distributed, nodal system such as this, it is necessary to define the individual processing elements, or nodes, and their inputs. To do so, the functionality of the nodes as well as the makeup of their neighborhoods must be addressed.

With the exception of the simple nodes that constitute the input and outut layers, each node in the system is a Gaussian classification node, so named because the fundamental component of a network node, the transfer function, is a Gaussian, or pseudo-Gaussian, function (Figure 7). As such, each node requires two defining characteristics: a covariance matrix, Σ, and a vector-valued mean $\underline{\mu}$. Figure 8 indicates the implementation of these defining characteristics. The covariance matrix for a node G_j is contained at the node, while the mean is defined as the n-tuple $<\mu_{1,j},\ldots,\mu_{n,j}>$ of afferent connection weights coming into G_j from the previous layer. These connections, then, define the neighborhood for G_j. For many applications, the node G_j will be connected to all of the nodes in the previous layer, though this is by no means a requirement. In fact, as will be seen in the vision application described below, much can be gained by restricting the inputs to a node to a subset of the previous layer.

Node G_j, so defined, is then quite simple in its functionality. Each processing element acts on its inputs at each time step according to the standard Gaussian function described in equation (19), identical to equation (2).

$$G_j(\underline{x}) = \frac{\exp\left(-0.5\left[(\underline{x}-\underline{\mu})^t \cdot \Sigma^{-1} \cdot (\underline{x}-\underline{\mu})\right]\right)}{(2\pi)^{(d/2)} * |\Sigma|^{(1/2)}} \tag{19}$$

Here $G_j(\underline{x})$ is the activation value of the jth Gaussian node when presented with vector input $\underline{x}$. Σ is the covariance matrix for Gaussian node G_j, while $\underline{\mu}$ is the vector-valued mean for this Gaussian. d is the dimensionality of this particular Gaussian and is equated with the number of input connections to node G_j. Since the components of the mean $\underline{\mu}$ are represented as a node's

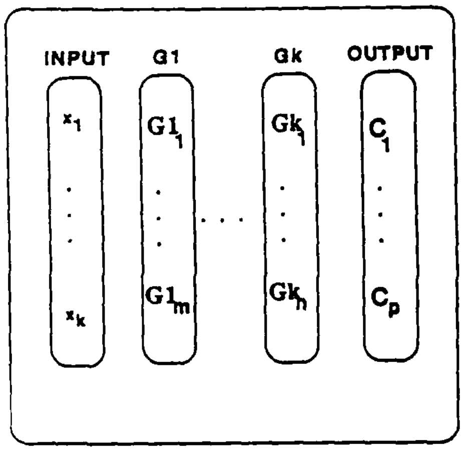

Figure 6. Layered Gaussian classification architecture.The layers G_1 are collections of Gaussian nodes connected to the preceding layer.

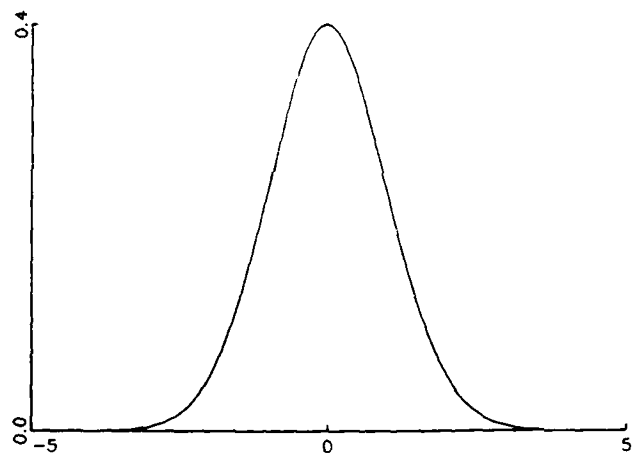

Figure 7. Gaussian distribution of mean 0 and variance 1.

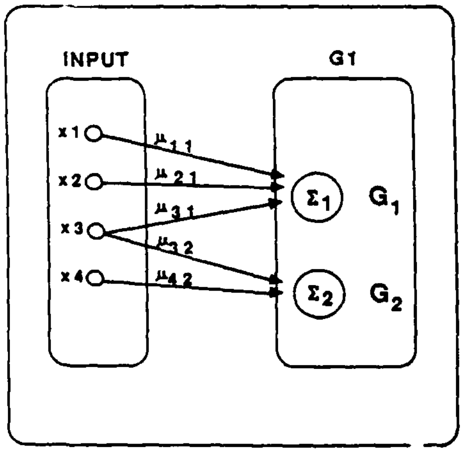

Figure 8. Diagram of Gaussian layer.

input weights, these weights can be thought of as shifting the origin of the input space for the node. This Gaussian function is then applied to the inputs of G_j, producing the node's activation value.

An important element of the system's power and flexibility stems from the fact that each individual Gaussian node defines its own covariance matrix, in addition to having a unique mean vector. This implies that the system ultimately consists of nodes whose activation functions, or transfer functions, are defined by the node itself. Each transfer function takes on its own unique shape, although all nodes retain the familiar bell-shaped Gaussian signature.

In practice, it is sometimes necessary to deviate from the strict Gaussian activation function and use a pseudo-Gaussian. By this we mean a transfer function retaining the basic shape shown in Figure 7, but not necessarily retaining other properties of the true Gaussian. For some applications the output of two or more nodes must be compared despite the inequality of their respective input dimensionalities. In these situations it may be necessary to alter the normalization term of the true Gaussian. A further discussion of these applications is undertaken in Section VI. The general form for the pseudo-Gaussian activation function is described in equation (20).

$$G_j(t+1) = \Delta G_j(t) + B_j * D_j * e^{-\eta E_j} \tag{20}$$

This equation gives the activation of Gaussian node G_j at time $t + 1$ in terms of the node's previous activation, $G_j(t)$, and an exponential function of E_j, the Mahalanobis distance between the input stimulus $\underline{x}$ and the mean $\underline{\mu}$ of Gaussian G_j defined in equation (21).

$$E_j = (\underline{x} - \underline{\mu})^t \cdot \Sigma^{-1} \cdot (\underline{x} - \underline{\mu}) \tag{21}$$

Δ is a short-term memory constant which allows an added dimension of history - the node's previous activation - to be incorporated into the activation calculation. η is a Gaussian parameter that alters the default standard deviation - the width of the bell-shaped curve - of the Gaussian function. B_j is a statistical parameter based on the number of patterns seen. This parameter can give an *a priori* estimate of the probability that the current stimulus belongs to the category represented by node G_j. In its simplest form, B_j equates to the ratio of the total number of patterns seen to the number of patterns that have been categorized as belonging to the category indicated by node G_j.

The nodes that make up the input and output layers are much simpler than the Gaussian nodes. For the input layer, we can think of each node as having a linear transfer function. These nodes simply send environmental inputs into the network unchanged. The output nodes, on the other hand, are simple summation nodes. These nodes perform the "sum of Gaussians" described above and output this result as the network's classification of the current input.

Network organization

The nodes should be thought of as organized in layers, similar to conventional neural networks. Specifically, we have three or more layers (Figure 6). The first, or input, layer consists of nodes having interactions with the environment. The nodes here are quite simple, and are primarily concerned with defining the computational problem for the ensuing time step. The input layer is therefore the bridge from the outside world into the network. It is through these nodes that the data is presented to the network.

The remaining layers, of which we need at least two, are essentially feature detection layers consisting of Gaussian classification nodes. It is in these feature detection Gaussian nodes, and in their afferent connections, that the learning takes place. The first Gaussian layer (G1) is made up of nodes with afferent connections coming from the input layer. An initial classification, or partitioning of the input space, is accomplished at G1. The first level of features, those explicitly present in the input pattern, are distinguished at G1.

The output layer is not actually a feature detection layer. At this layer the network performs a "combination of Gaussians" (Figure 9). This technique, taken from kernel estimation theory (17), allows the network enormous flexibility. The nodes in the output layer correspond one-to-one to the classes being discriminated. The non-zero afferent connections into an output node O_j from layer G1 indicate the features, or preliminary Gaussian classifications, that are to be combined to make up the ultimate distribution for class C_j. While each of the G1 nodes that are summed to create O_j are themselves Gaussians, the final distribution need not be constrained to be a Gaussian. As described above, a wide variety of distributions can be approximated by a mixture of Gaussians. The network has the flexibility to model a particular class C_j based on the statistics of the input the system has seen, rather than being forced to fit the data to a Gaussian.

The input layer, one or more Gaussian feature detection layers, and the output layer comprise the network. Upon receiving an input stimulus (visual pattern or digitized waveform, for example), the information propagates through the network, driven by local nodal calculations, ultimately determining the activation values of the output nodes, and hence the individual class

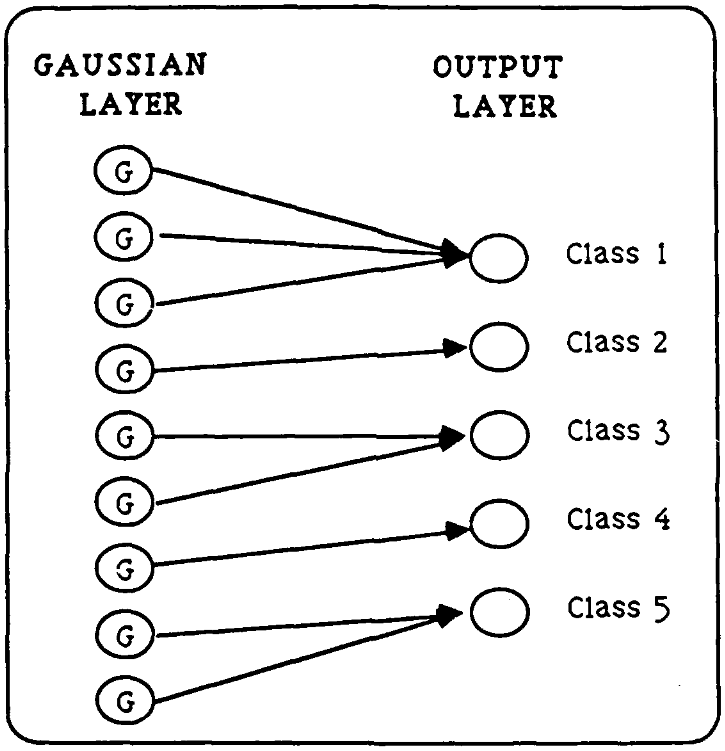

Figure 9. Diagram of classification layer.

values. These outputs are then interpreted as the relative likelihoods that the current input belongs to the individual classes. Figure 10 gives examples of interpretations of network outputs.

The network's output values yield a dual interpretation. First, the relative values are important. The output node with the highest activation value corresponds to the class to which the network determines the input belongs. In Figure 10a, node O_2 is the obvious winner. In addition to this relative interpretation, there is an absolute one. In Figure 10b node O_1 has the highest value and is therefore the network's best guess as to the classification of the input. However, the low value of this winning node indicates a lack of confidence in this classification. It is more likely that the input stimulus used to evoke this output belongs to an entirely new class -- a class for which the network has not developed a representation. Similarly, in Figure 10c, two unique classes are indicated with high likelihood. This implies that, although one of the nodes (O_3) has the highest value, both classes C_2 and C_3 are potentially correct classifications for the input. In other words, classes C_2 and C_3 in Figure 10c have some overlap, and the current input falls in this area of overlap.

Operating environments in which there is a penalty assessed for misclassification, such as medical diagnosis, must have this absolute meaning inherent in their outputs. If, for example, Figure 10c depicts a situation in which failing to recognize the high likelihood of class C_2 could lead to a fatal misclassification, it is imperative to indicate class C_2 is potentially the correct classification.

It is now possible to see that each node is independent of the other nodes on the same layer. This implies that all the nodes on a given layer, say G1, can process in parallel. Since a layered system such as that depicted in Figure 6 can be thought of as a pipeline processing architecture, the entire system, when realized in parallel hardware, can accept inputs with a relatively fine discrete time step. The delay between successive inputs need be no longer than the time it takes to calculate a multi-dimensional Gaussian function. The slowest individual node in the system defines the time step, and that node need perform only a single, simple function calculation.

Learning techniques

While the information flow through the network defines the classification architecture, it is still necessary to describe the rules for network adaptation. It is unacceptable to require that the system be handcrafted for a particular problem. There must be a method allowing the network to adapt dynamically to incoming stimuli, thereby developing an internal representation for the

Interpretation of Network Outputs

(a)	(b)	(c)
$O_1 == 0.17$	$O_1 == 0.23$	$O_1 == 0.19$
$O_2 == 0.67$	$O_2 == 0.11$	$O_2 == 0.83$
$O_3 == 0.24$	$O_3 == 0.13$	$O_3 == 0.87$

Figure 10. Example outputs from a 3-class system.

statistics of the stimuli. This internal representation is necessarily created from layers of Gaussian feature detectors and the eventual summation of Gaussians, as described above.

This adaptation is necessary because of the restrictions on processing required by a given application. While the kernel estimator may in some instances perform better, it may require far more "nodes" than are available in hardware. Another consideration is the problem involved when the input dimension is increased. A kernel estimator may require an extremely large amount of teaching data in higher dimensions [13]. The hope is that the adaptive system will function well with fewer points, and experiments with applications seem to bear this out.

The adaptation of the network takes place in two separate and distinct forms: alteration of the Gaussian means $\underline{\mu}$, represented as connection weights, and alteration of the covariance matrices Σ contained in the individual nodes. These two adaptation techniques allow the network to develop a useful representation of the data. The adaptation of the covariance matrix for a particular node G_i, described in equation (15), allows the network to reach a state in which every node potentially has a unique activation function. The weight updates described in equation (9), altering the means for each Gaussian, provide for the modeling of the data. These adaptation techniques allow for the creation of a distributed, redundant model of the data to be used in classification.

There are two classes of learning, or adaptation schemes. The first, unsupervised learning, requires the system to develop its representations independent of any feedback as to the correct classification. The system is presented with a sequence of multi-dimensional input stimuli

$$\underline{I} = (i_0, i_1, \dots, i_n) \tag{22}$$

and is required to partition these inputs into categories. In other words, in unsupervised learning the network is never told what the stimuli entering the system represent in the environment. As such, it is inherently difficult to get a clear picture of the environment. The best the network can hope for is to cluster the inputs into categories based on their fundamental characteristics. Overlapping or confused data will necessarily be clustered in the same category.

In supervised learning, on the other hand, it is possible for the network to gain a detailed understanding of the environment. Supervised learning requires, however, that the network have access to ground truth data (teaching data). Feedback as to what the particular inputs actually represent must be fed into the system, thereby allowing the network to build its representation in an informed manner. The network is presented a sequence of teaching pairs

$$\mathbf{I} = ((i_0,C_0), (i_1,C_1), \ldots , (i_n,C_n)) \tag{23}$$

Each pair consists of an input stimulus and the correct classification of that stimulus. In this way the network can build a representation that clarifies some of the details of a confused environment. Although it is still impossible to determine the proper class in an overlapping category situation, it is possible to make educated guesses based on the probabilistic information content of the teaching data. An important point here is that the teaching data must be representative of the input the network is going to see in order to derive optimal performance from the network.

In many application environments a combination of supervised and unsupervised learning may be best. Initially, the network's internal representation is developed using a teaching data set for which the truth is known. After this initial learning phase, the system is placed in the operational environment and allowed to adapt in an unsupervised manner, altering its representation if necessary. The network, then, receives an input sequence that is the concatenation of a teaching set with an unsupervised input sequence. Numerous situations arise in which adaptation beyond the initial supervised learning stage is necessary. Instances where the teaching data may not be entirely representative of the actual data the network will receive may include distributions where the known classes may change slowly in time (drift), or when a new class appears in the input data that was not represented in $\mathbf{I}$.

The learning rule consists of two parts. First, each node must decide whether or not to coalesce the new input into itself, using the rules for moving mean and moving covariance described in Section III. The network must then decide whether the point is adequately covered by the existing nodes, or whether a new node should be allocated.

The tradeoff described in Section II between too few nodes per class and too many nodes per class has a middle ground wherein the actual distribution is closely approximated. In practice, a network has finite memory and processor resources. In the three layer network paradigm, there are only N Gaussian nodes available in layer G1. It is necessary to develop an internal representation which not only models the data, but does so within the physical constraints. To do so there must be a dynamic criterion for making the decision between allocating a new node for input i_k and coalescing i_k with one or more previously created Gaussian nodes. The most obvious technique for making this decision is to choose a value **c** as the window for inclusion. Let the unscaled value of the node be defined by

$$\underline{G}_q(i_k) = (2\pi)^{(d/2)} * |\Sigma|^{(1/2)} * G_q(i_k) \tag{24}.$$

If input stimulus i_k enters the system, and, for all nodes G_q of the same class as the input, we have

$$\underline{G}_q(i_k) < \mathbf{c} \tag{25}$$

then we will create a new node with mean $\underline{\mu} = i_k$ and covariance equal to some initial value Σ_0. In the unsupervised case, we create a node if all nodes so far satisfy (25), regardless of the class associated with them.

To decide whether to coalesce the input into a given node, we need another constant **u<=c**. If for some node G_q we have

$$\underline{G}_q(i_k) \geq \mathbf{u} \tag{26}$$

then we coalesce the input i_k into the node G_q using the moving mean and covariance formulas (9) and (15). In the case of supervised learning, the node is required to be of the same class as the teaching input in order to be updated. In the unsupervised case, all nodes are updated, proportional to the output for the node's class.

The window values **c** and **u** should not be static. As mentioned above, hardware constraints necessitate a dynamic window. The Gaussian layer must monitor itself, dynamically altering the window value to ensure that the system does not allocate too many nodes. There are a number of ways to approach this problem, and it is the focus of ongoing research.

The method of network adaptation described is two-fold: adapting the mean and adapting the covariance. The two can be separated for discussion, and in fact, many application areas do not necessarily require the adaptive covariance, and in others it might be preferable to adapt the covariance but leave the means fixed.

The first, and simplest, learning scheme involves using the moving mean technique while requiring the covariance to remain fixed. This method can be thought of as simply creating Gaussian nodes according to equation (24) and adjusting their means according to equation (9). A partitioning of input space by hyperspheres of equal size is accomplished. When classifying a novel input stimulus in this scheme, the network essentially ranks the classes by distances from the input point.

Adapting the covariance along with the mean allows the system more flexibility. The moving covariance (15) allows the additional ability to alter the size of the individual hyperspheres independently, based on the statistics of the incoming data. This then yields a field of Gaussian hyperspheres of varying size, and the network's classification technique is to have each Gaussian output its value when evaluated at the current input i_k.

An important feature of the proposed learning algorithms is their one-pass nature. If this system, or any other system, is to be called a truly adaptive network, then it is somehow incongruous to require off-line learning. Such a learning scheme is more properly thought of as static, with an entirely new network replacing the old one at intervals, i.e. when a new network has been taught on the most recent teaching data available. On-line learning, on the other hand, requires an algorithm that can perform in real-time or near real-time. Although this criterion is dependent on the environment to some extent, it is clear that real-time learning implies a single-pass, or at worst a few-pass, learning algorithm. The scheme described herein is a one-pass algorithm. There is no need to present a particular input stimulus to the network repeatedly. The learning consists of updating the weights (means) and the nodal transfer functions (covariances), and allowing the system to adapt to the newest input data i_k at the same time the information is flowing through the network. There may be times when it would be beneficial to have a static network, a network that will not adapt to incoming data after an initial learning period. The adaptive Gaussian network can accommodate this need. However, it is in the arena of truly adaptive systems that the Gaussian network shines. Here, real-time learning is a reality.

Figure 11 shows the results of running 500 points of simulated data taken from a Cauchy distribution through three systems: a kernel estimator (KE), a fixed-mean, moving-covariance estimator (FMMCE), and a moving-mean, moving-covariance estimator (MMMCE). Note that the FMMCE is comparable to the kernel estimator, but uses fewer nodes (recall that the kernel estimator requires one Gaussian node for each point). If the means are allowed to vary, the Gaussians will move toward the nearest mode, causing the modes to become accentuated. This can be undesirable in a density estimator, but it can be useful for many classification tasks. In a classification task, the "features" are the modes of the distribution, and the classifier must be able to distinguish between these features. In fact, in many cases, as seen in the applications in Section VI, the covariance can be fixed, remaining unvarying throughout learning. In these systems, the network learns to find the modes of the distributions; it learns the features which distinguish the classes.

In the example given in Figure 3, it was noted that a problem can arise if only the value of the distribution is used in classification. In that example, a point near the mean of the distribution for high risk patients could be mistakenly classed as a low risk patient due to the fact that it is also near a mode of the low risk distribution. This problem can be remedied by returning a distance from the mode, rather than the value of the distribution at the mode. This can be accomplished

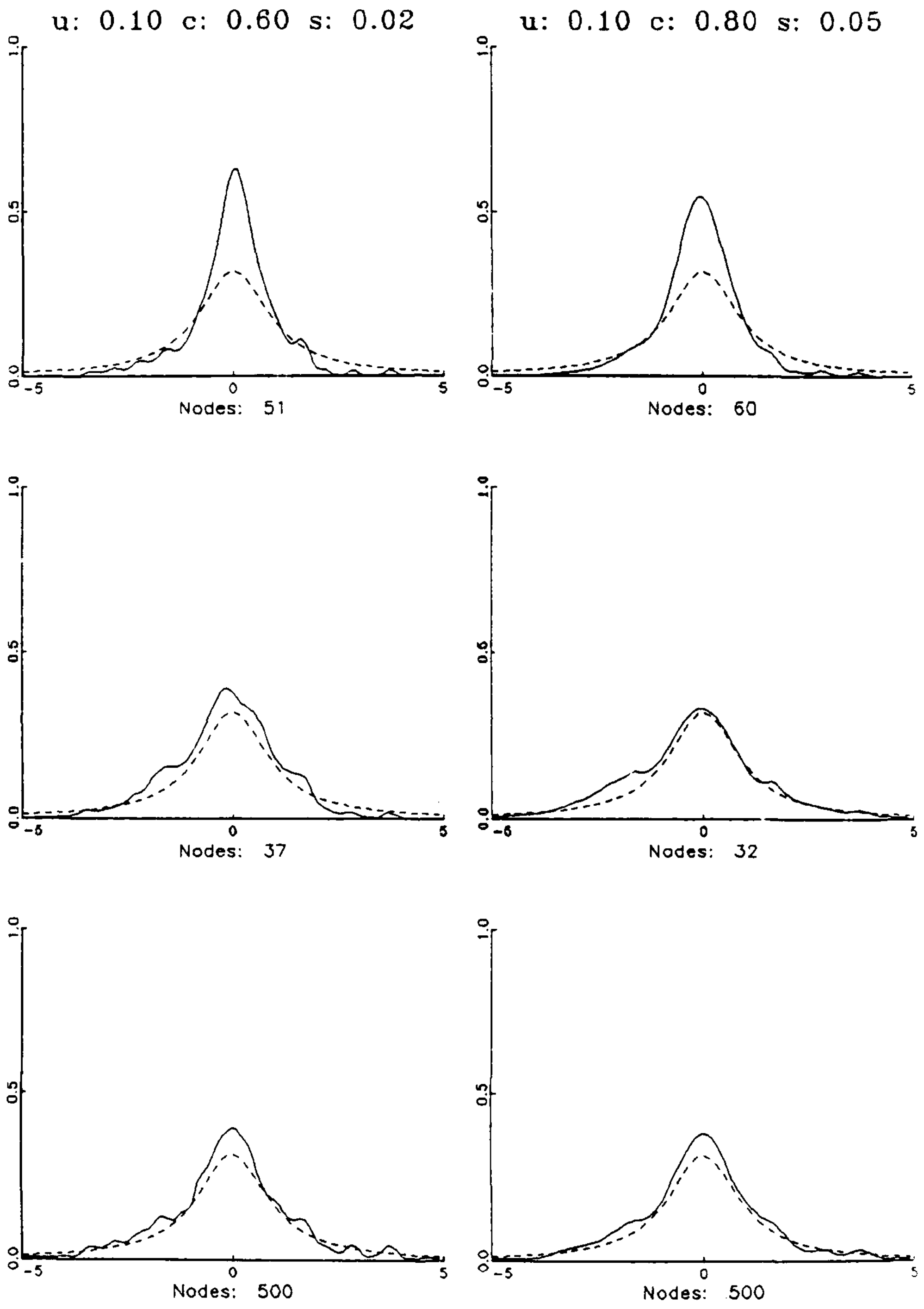

Figure 11. Density estimates using 3 techniques. The data used for the estimates was sampled from a Cauchy distribution (dotted curve). Each column shows the three techniques run on the same data with the initial constants as listed at the top: u = update threshold; c = create threshold; s = starting variance. The techniques are top: moving mean moving variance; middle: fixed mean moving variance; bottom: kernel.

using a moving mean architecture since, as seen in Figure 11, the nodes tend to cluster around the modes in this case.

VI. Applications

The network architecture described in Section V, with slight modifications dictated by the particular application, is currently being used in a variety of applications. Among these are isolated word recognition, feature detection in vision, sensor fusion in a robotic environment, interpreting magnetic resonance inputs and parametric classification of clouds. We will describe three of these applications: sensor fusion, isolated word recognition and vision. The network runs in the C programming language on a variety of computers, including SUN, VAX, Convex mini supercomputer and PC-AT. For the most part, due to the relatively fast learning allowed by the one-pass adaptive mean/covariance techniques, the applications can be performed on an 80286 based microprocessor.

Robotic sensor fusion / sequence recognition

The ability to understand one's environment is not governed solely by static pattern recognition. The order in which events occur can be even more important than the events themselves, and an intelligent system, whether it be a mouse or a robot, must be able to detect and understand this ordering. Thus the dimension of time allows access to a wealth of information about the current environment, past events, and expectations about the future. An ability to incorporate time into information processing is necessary for abilities such as the recognition of sequences of events, understanding cause and effect, and making predictions and planning.

The ability to recognize sequences is essential for many tasks, most notably those involved with audition and vision. A sequence may consist of a stream of phonemes, typed letters or frames from a movie. Once the initial preprocessing has been done and the individual members of the sequence have been recognized, the task is shifted. The processing is then concerned with determining which of the known sequences are represented by the input. The answer may depend on the context in which the input has been received. The system therefore must return all the sequences that the input might represent, with a confidence rating indicating the quality of match between the input and the known sequences.

Consider a stream of phonemes. The task is to recognize the words that are being spoken. Here it is important to recognize the order in which the phonemes appear and the words these sequences of phonemes might represent. A certain amount of error will appear in the sample due to normal fluctuations in a speaker's voice and a large number of variable conditions in the environment. Also, a sub-sequence may be a legitimate word which the system should recognize in order to allow a more sophisticated system to deal with the ambiguities.

A simple formulation of this recognition problem, similar to that provided by Tank & Hopfield[14], is shown in Figure 12. The problem is to extract known sequences from noisy data. Ideally, we must be able to recognize the sequence "I D A H O" from the stream given, despite the fact that a perfect match is not present. The letters in the figure can be thought of as abstract events such as sensor firings and the words as higher-level activities, or sequences of events. The letters along the x-axis are the inputs to the network, received in the order shown (left to right). After each input, the network attempts to correlate the input received thusfar with the sequences the system has been taught (**iowa, idaho, utah** and **ohio**). The y-axis (Recognition Results) indicates the degree of match between the known patterns and the input stimuli. These values are the outputs of the Gaussian classification nodes corresponding to each known pattern. The fact that **ohio** and **idaho** are present in the input, albeit with noise, is indicated by the relatively high values output by the classification nodes corresponding to these sequences. All the necessary characters for the sequence **iowa** are present in the input, but they appear in improper order. This is indicated by the lower, but non-negligible, output from the **iowa** classificaiton node. Each known pattern has a space-character as the last character in the pattern to aid in determining word breaks. This character is also present as the last input stimulus.

To process this stream of events the system must be able to represent order information and work on imperfect exemplars. The duration of the constituents of the sequence and the spacing between them is also important. The system must be able to incorporate this information, learn the sequences, and adapt to the environment in which it is operating.

The network used for robotic activity recognition runs on an 80386-based microprocessor, and is very similar to the architecture described above. The application described here uses ROBART II [3], a mobile sentry robot resident at Naval Ocean Systems Center, as the testbed. For this application we think of the robot as a sensor platform. ROBART II has multiple homogeneous and heterogeneous binary sensors, and the task is to accept input from these diverse sensors, fuse the incoming data, and develop an understanding of the activities taking place in the environment. To do this, the Gaussian network must be able to accept the various data into a single field, and develop a representation of actvities of interest that incorporates time as well as sensor type. The robot has many diverse sensor types, but these initial results are based on just two working sensor sets: one microwave sensor and five infrared sensors.

The architecture consists of three layers. The input layer is temporally ordered using a decay scheme. The Gaussian layer recognizes sequences of events, and the output layer sums the

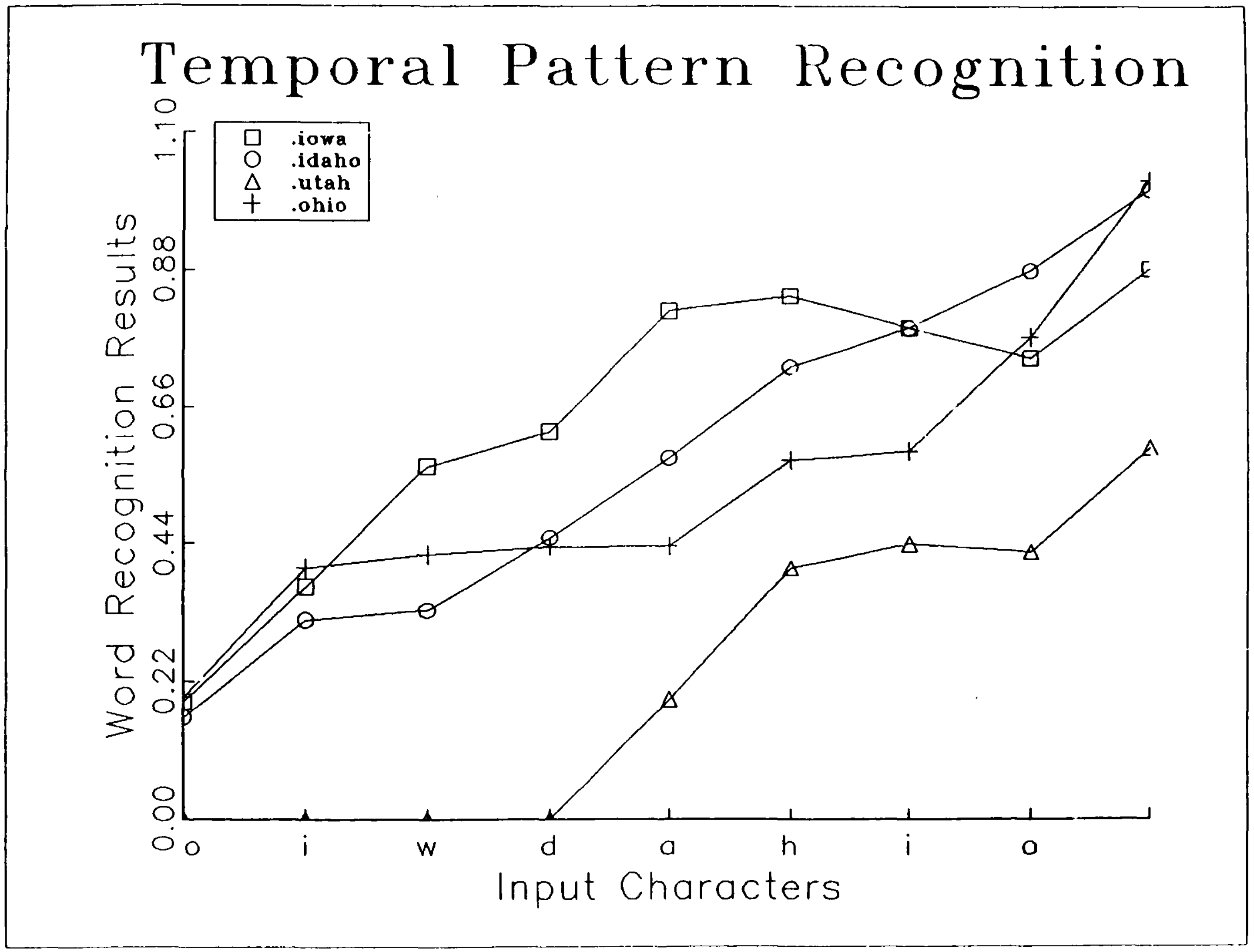

Figure 12. Output of word recognition system as a function of input.

output of one or more Gaussians (Figure 13). Data comes from the robot in sensor packets, individual information packets indicating which sensor fired. Ultimately, the network will be resident on the robot, or on a computer with a link to the robot.

The temporal ordering scheme used is straightforward. After input stimuli are received into the input field from the various sensors, the output values in the field experience a decay factor that acts as an ordering function [5],[9]. By using this decay (as seen in Figure 14) the input field can develop a representation in which the order the stimuli were received is preserved. This decay is implemented via the following formula:

$$a_j(t+1) = a_j(t) - \Gamma * a_j(t) \qquad (27)$$

where $a_j(t)$ is the activation value at time t for input node j, and Γ is the decay rate. Γ is chosen large enough to sufficiently separate the inputs for the Gaussian classification. This ordering can be used to distinguish between two or more sequences having the same elements, but in a different order. Since a different order will produce a different pattern on the input layer, the Gaussians can distinguish between different sequences. The distance of the current input field from the learned patterns already represented on the Gaussian level is the basis for determining the degree of match, and therefore the network's classification of the current input.

It is clear that the input field must contain more than a single node for each distinct stimuli. For example, if the system is categorizing sequences of letters, there must be more than one input node corresponding to the letter "A". Were this not the case, the second "A" entering the system would activate the only input node corresponding to "A", overwriting information concerning the previous "A".

Evaluation of the performance of any temporal pattern recognition system is far from a straightforward task. In many, if not most, instances the incoming data does not fit exactly with any of the learned sequences. There is little value in a yes/no decision on the presence of a sequence. The system is asked instead to give a "best guess" of which pattern(s) it is seeing based on some set of criteria. This is by definition an ambiguous task, and the ultimate result can only be evaluated by looking at the criteria and the input data and attempting to generate a "better" solution, as defined by the human evaluator.

The network has been tested on simulated data under a variety of environmental conditions. It is necessary to recognize sequences of sensor firings as one of a set of learned sequences even if the incoming unknown sequence contains extraneous firings, is missing elements (noise conditions), or multiple sequences are occurring simultaneously.

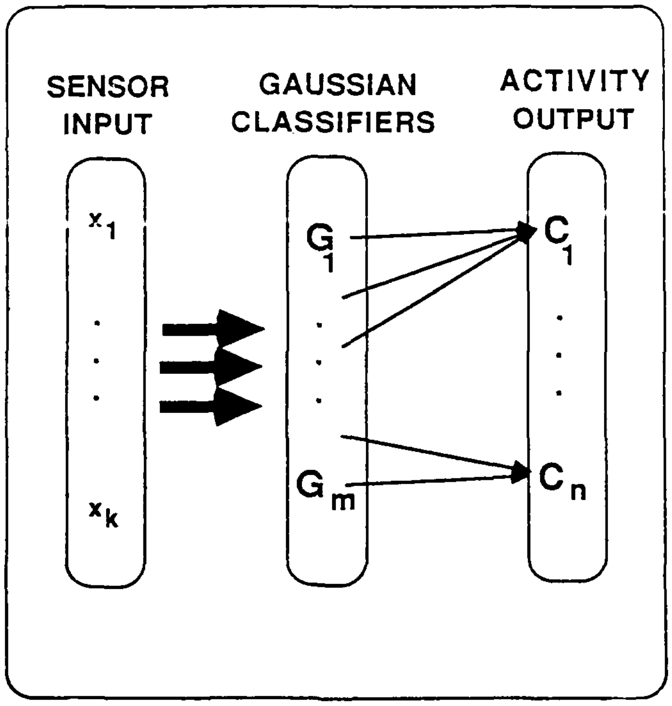

Figure 13. Architecture used for robot activity recognition.

Decay at F3 level: $\Gamma = 0.15$

Input : A B C D E

INPUT LAYER

A	= 0.5220
B	= 0.6141
C	= 0.7225
D	= 0.8500
E	= 1.0000

Figure 14. Example of decay after 5 inputs to the system.

The simplest and most straightforward experiment that can be used to test the quality of the system is that of sequence recognition in a noiseless environment. Here it can be determined whether the system has performed properly or improperly. As a complete sequence is input it exactly matches one of the learned patterns, and it is imperative that the system correctly identify this pattern. In addition, the system must exhibit the ability to indicate the presence of multiple patterns. Multiple patterns are ultimately flagged as present due to the relative magnitude of their activations compared to the activations of the other patterns stored in the system. This is vital since many robotic environments do not ensure mutually exclusive events or patterns. The system may give as output more than one pattern that has a high value. This represents the fact that the Gaussian nodes corresponding to each of the indicated patterns received input close to their corresponding means, and therefore the activation values for these nodes are close to the maximum possible value the Gaussian can attain.

In evaluating the system under different stimuli conditions, it is useful to understand and assess the prediction ability of the system. Prediction is the ability to indicate the presence of a pattern prior to receiving the entire pattern. If the network has been taught that a particular sequence of twenty sensor firings indicates an intruder has entered the room, we expect the network to recognize as relevant the sub-sequence made up of the first fifteen of these sensor firings. A system with little or no prediction capability is of limited use, since the system user would like to be warned of possible happenings prior to their conclusion, thus allowing the user to affect the ultimate outcome by acting on early stimuli. The prediction capabilities are relevant in both noisy and noiseless environments.

Data may be "missed" due to sensor inadequacies or by subtle variations in the actual pattern itself which alter the sequence in some small way, yet leave the overall meaning of the pattern unaltered. Although the pattern received does not correlate exactly with the learned pattern, the network must indicate that the pattern of input stimuli is similar to one of the learned patterns. There is no hard and fast rule for determining how certain the system should be that it is seeing a given pattern. The only definitive statement that can be made is that the system must give some indication that it is close to seeing the learned pattern. The adaptive Gaussian system actually outputs a "distance" from the learned Gaussian mean to the received stimuli, using this as its analog output. Therefore the system indicates the presence of a particular pattern despite the fact that the input stimuli consists of only a partial pattern. An intruder entering the room is indicated with some smaller degree of certainty despite the absence of two sensor firings that the system has been taught belong in the pattern.

In the case of extraneous data, or noise, the network is asked to ignore the noise where possible and process only the relevent information to determine the existence of learned patterns.

The individual Gaussians are concerned only with inputs that are represented in their dimensionality, and therefore ignore stimuli that are not a part of the Gaussian domain. For stimuli present in the Gaussian dimensionality, the Gaussian nodes attempt to determine which input of a particular class (or dimension) is closest to the component of the mean for that dimension. In this way, extraneous data are disregarded.

The system can be altered to perform at various stages along a continuum from order being all-important to disregarding order. The extent to which the system can identify patterns despite the stimuli being received out-of-order rests on a design decision tightly tied to the type of environment in which the system operates. Since there exist no generic criteria for determining how important ordering should be, the network needs to be flexible in this regard.

Performance of the system on sensor data has been analyzed. The network was taught to recognize two separate scenarios: someone entering the room and someone exiting the room. The similarity in these scenarios figured in their selection for testing purposes. In a general sense, one is just the temporal inverse of the other. The sensors that fire when a person enters the room should be, for the most part, identical to the sensors that fire upon exiting. Roughly speaking, the major difference is that particular sensors fire in the reverse order. As such, these scenarios force the system to actually incorporate the ordering of the sensor firings, as well as which sensor fired, in order to discriminate between the two.

The network is taught via supervised learning, using multiple instances of a scenario. Four separate people were observed entering the room, and the sensor firings that occurred during these activities were fed to the net along with feedback that the activity "entering" was occurring. The four "entering" scenarios are then coalesced, using the Gaussian learning scheme described above, into a template for the activity "entering". Similar teaching sequences were used for the "exiting" scenario.

At this point the network has developed a representation for both activities, and can be tested. Sensor firing data from the robot is collected over a long period of time, and this data is fed into the network. It is known where in the data certain activities occur, but the network is not given this information. The activation values for the network's output nodes are traced, and the network does in fact indicate the presence of the activities at the proper points in time. Figure 15 indicates this recognition. The x-axis indicates time, from right to left. As time progresses, the network receives sensor firings from the robot. The y-axis indicates the output of the Gaussian classification nodes, with each graph depicting the output of a seperate node. A spike in the graph is indicative of the network correlating the most recent sensor firings with a known pattern. Notice the fact that when the network indicates the presence of "entering", it also sees "exiting", albeit with a lower likelihood. This is expected, and arises from the inverse-time relationship of the two

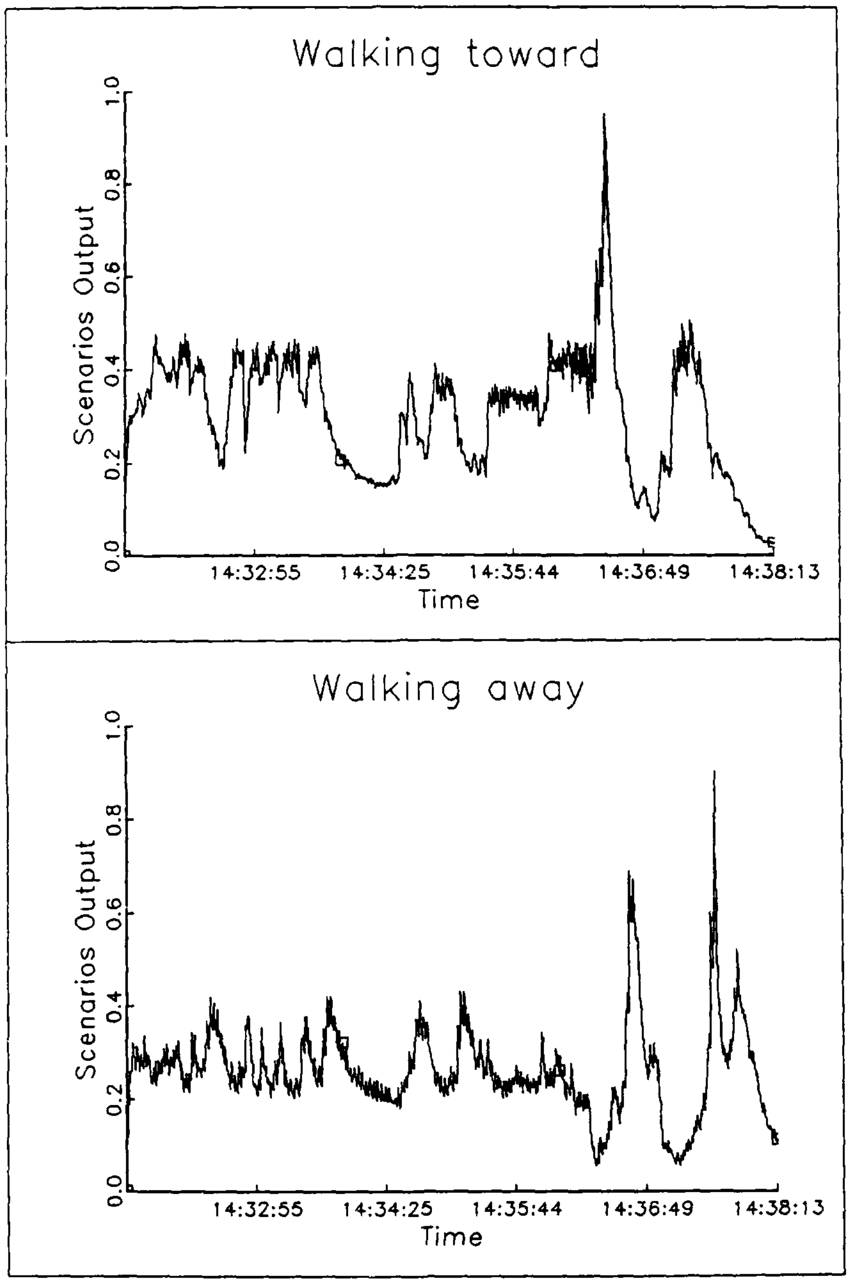

Figure 15. Output of robot activity recognition system. The top graph is the output of the node corresponding to the event "person walking toward robot" and the bottom is the output of the node corresponding to the event "person walking away from the robot." During the time shown a person entered, approached the robot, then retreated and exited. Spurious sensor firings, inherent in a complex environment, are the cause of the high noise level seen.

scenarios.

The data used in Figure 15 was collected using two sensor suites: IR and microwave. The microwave sensor was malfunctioning, causing it to fire at any motion, and this resulted in a high level of noise. This noise is the cause of the high output values for the network when "nothing" is happening. Since in this data set the scenarios consist of the firing of three of the IR sensors and the microwave, the microwave comprises one fourth of the required sensors. As a result, it has an exaggerated importance in the network. Even with this handicap, the network can still correctly distinguish between the two scenarios and learns to ignore the spurious firings.

Isolated word recognition

A promising application of the adaptive Gaussian network scheme is in the area of speech recognition. Specifically, we have applied a variation of the network to the problem of recognizing isolated words. The system currently runs on an 80286-based microprocessor. The input to the system is the digitized waveform (amplitude-time domain) output by a coprocessor A/D board in the computer. Sampled at 8KHz, integers representing the amplitude of the signal are fed into the system. The network architecture employed is shown in Figure 16.

A speech signal is a nonstationary signal, variable in the time domain. As such, it is necessary to break such a signal into smaller wave segments, each of which approximates a stationary signal. From this approximated stationary signal, one can extract information and analyze the waveform. An example of this technique is the short-time Fourier transform. The parameters of a speech signal (the frequencies present and the transitions between frequencies) are changing throughout the duration of the signal, and it is these changes in time that must be exploited in the recognition of the waveform.

For our purposes, digitized waveforms representing individual utterances of the digits 0 through 9 were recognized. Gaussian classification with moving mean is performed on the amplitude-time domain representation of the utterances. The results cited below are for a system in which the covariance is kept constant. Analysis of the data indicates, however, that an adaptive covariance scheme will increase the recognition abilities of the system. Research is currently ongoing in this area.

This system uses a temporal representation combining energy, zero-axis crossings of the signal and zero-axis crossings of higher derivatives of the signal, and uses the temporal reliability of these parameters to classify an unknown utterance as one of a known set of learned utterances -- in this case the learning consists of the ten digits 0 through 9. Kedem's work [6] shows that higher-order zero-axis crossing statistics can be very useful as discriminators. A Gaussian

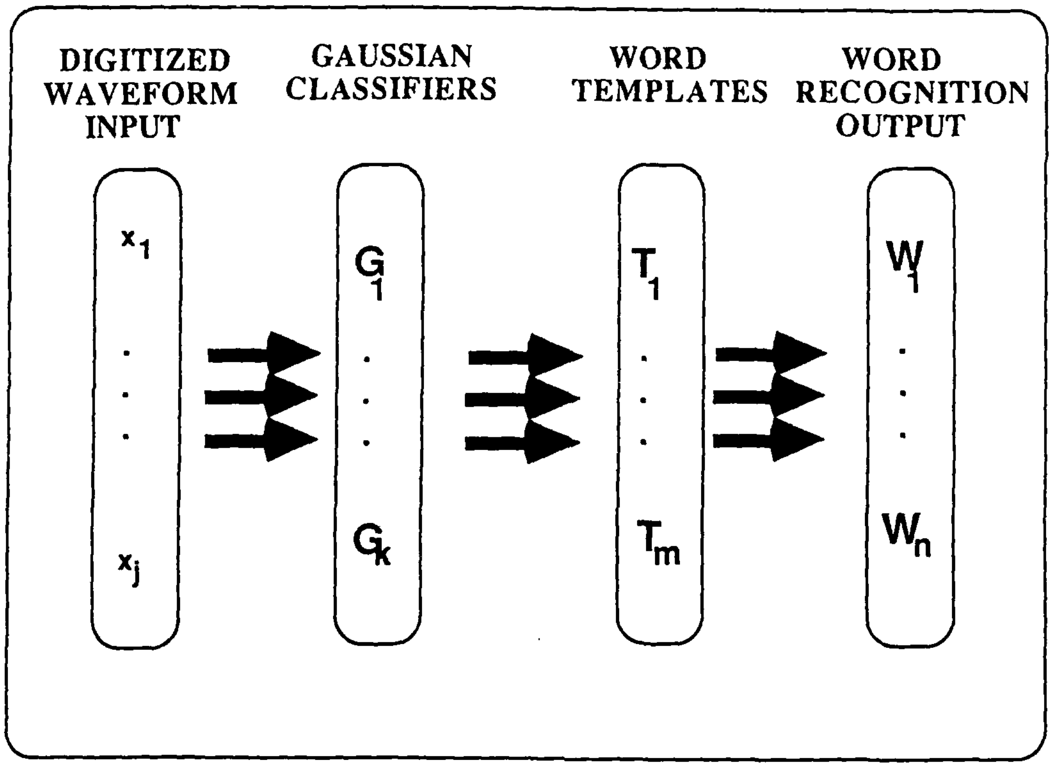

Figure 16. Speech recognition architecture.

template for "ONE", obtained through the coalescing of numerous utterances of "ONE", consists of multiple multi-dimensional Gaussians (Figure 17). The Gaussians that make up an utterance template serve different purposes. The firs., the Z_0-Gaussian, is a temporal representation of the zero-crossings within the utterance. The information contained in the Z_0-Gaussians is predominantly low frequency information. The Z_1-Gaussians, on the other hand, contain information about the zero-axis crossings of the first derivative of the signal (peaks and valleys) throughout the utterance. These crossings contain higher frequency information about the utterance. Each successive derivative of the signal gives rise to a Z_i-Gaussian which is tuned to higher frequencies than the preceeding Z_{i-1}-Gaussian. There is, of course, a limit to the number of derivatives that yield information, and hence that have discriminatory power. We have found, however, that there exists important information for speech recognition through at least the sixth derivative.

These Gaussians are then combined to yield the ultimate template for a word (Figure 17). In addition to the zero-axis crossing Gaussians, an E-Gaussian representing energy information is included in the template. Once the system has been taught to recognize the required words, an input stimulus representing an unknown utterance returns a Gaussian distance from both the Z_i-Gaussians for each template. These distances are combined to give the final output of the network -- the relative liklihood that the input stimulus belongs to each of the learned categories.

Each dimension of these multi-dimensional Gaussians is a small portion of the entire utterance. Dividing the utterances into sections, each of whose length is a function of the total length of the digitized utterance, provides a straightforward method of length-normalization which is useful in the Gaussian classification stage in addition to yielding an approximation of a stationary signal. If we let N be the desired dimensionality of the Gaussians, then it becomes necessary to divide the utterances into N sections. Let L be the length of an utterance in digitized samples. Then each section would be L/N samples long. However, since it is desirable to have overlapping sections, we want

$$K = L / ((N + 1) / 2) \tag{28}$$

where K is the length of each section.

Then, rather than use this simple count of the number of occurances of zero-axis crossings in a section, we use this count divided by the length of each section, K, to obtain the frequency of occurance. In this way we implicitly time warp the utterances such that the impact of differences in

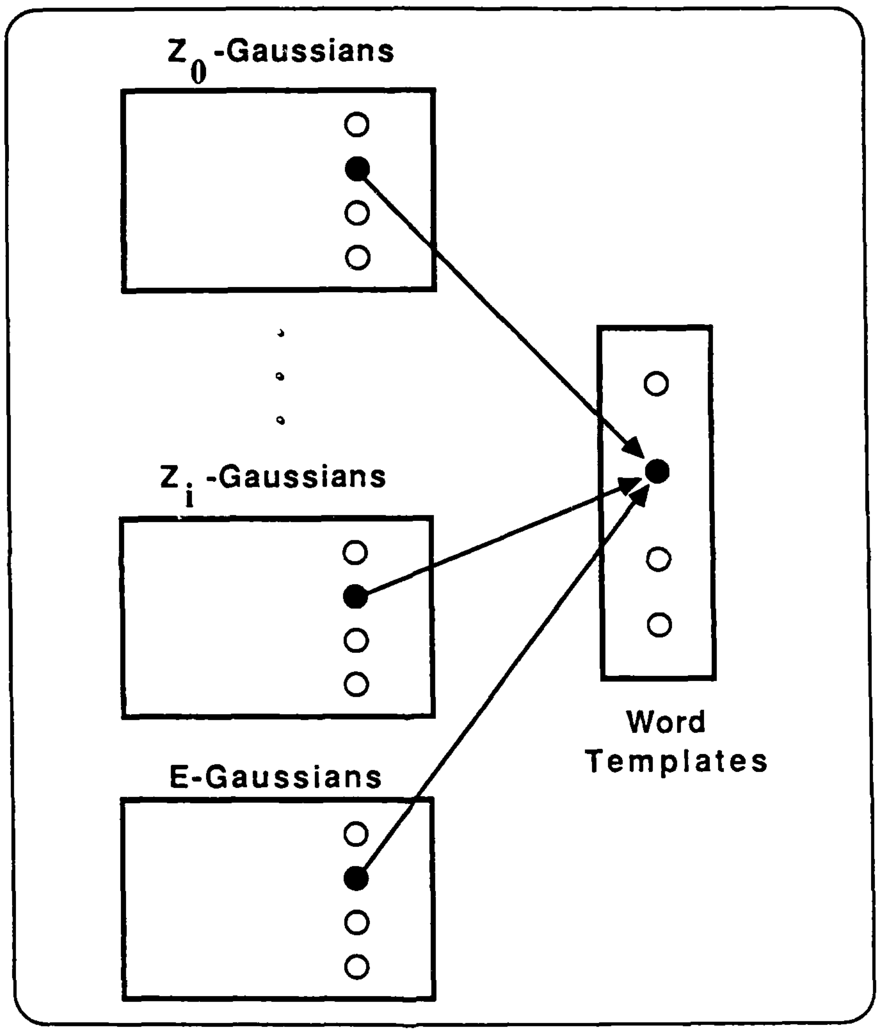

Figure 17. Word templates for speech recognition system. Each template consists of the output of several Gaussian clas. fiers.

length is minimized as long as the relative frequencies of these two parameters remain similar.

By taking ordered, overlapping counts of the frequency of occurance of both zero-crossings and peaks, the system is able to characterize an utterance according to the Gaussian distance from some teaching mean for each category, established using the moving mean calculation from equation (9). The representation at the input level is m+2 separate N-dimensional fields; a field for the energy, a field for the zero-axis crossings of the signal itself, and m fields representing higher-derivative zero-axis crossings, where m is the number of derivatives. As the input stimulus enters the system, the input level breaks it into sections as described above. For each section, a count of the number of zero-crossings (or an energy calculation) is obtained, and this count is then divided by the length of the section to obtain a value for the frequency of zero-crossings in that section. This value is entered in the arrays at the input level, and the procedure continues for the next section of the current input. When the input waveform has been completely received by the input field, all N sections will have calculated a frequency value and deposited this value into the appropriate slot in the input arrays, yielding an m+2 by N dimensional input array representing the temporal zero-crossing and energy information inherent in the signal.

The Gaussian field then uses the input arrays to create the Gaussians to be used as templates in the recognition of novel input stimuli. If a template already exists for a word when new training data enters the system, the input arrays are coalesced with the existing Gaussians via the moving mean calculation to obtain a refined word template.

These Gaussian categorizations, thus coalesced, become the templates for the classification of unknown utterances. In this case, input arrays are compared with the Gaussian templates to obtain a Gaussian distance from each stored word template. These distances indicate which category the system believes the input stimulus comes from. The system performs quite well on the limited vocabulary tested, with correct categorization in the 90% - 100% range for speaker dependent recognition. Table 1 summarizes the results of this word recognition. There are five sets of utterances, a through e. Each set consists of one utterance from each of the ten digits. The network is taught on four of these sets, and the remaining set is tested to determine network performance. These results compares favorably with conventional approaches [10]. The system can also be used to perform speaker independent recognition, with many varied utterances from different speakers being coalesced into one or more Gaussian templates for each word.

This scheme for individual word recognition is conceptually extensible to continuous speech. Since the Gaussians that make up the word-templates independently calculate their distance from the input field, it is possible to consider this an ongoing process, performed at each time interval. In this way, the distortion of words due to run-together will cause the system performance to degrade, but the mechanics of the processing will remain intact. In fact, if the

Table 1. Results of speech recognition system on isolated digits.

TEACH	TEST	CLASSIFICATION RATE
a,b,c,d	e	90%
a,b,c,e	d	90%
a,b,d,e	c	100%
a,c,d,e	b	100%
b,c,d,e	a	100%
overall		96%

system were taught on input data of this type (containing run-together), the performance would increase. This may, however, require a more intensive training period. Also, the processing requirements for such a system would be quite intensive if the system is considered as a sequential simulation. As a parallel system, these requirements would not be overwhelming.

Vision

Preliminary work has been done on a system to recognize objects, in this case hand written characters, under conditions of translation, noise, and scaling. The system is required to recognize which characters are in the image, and indicate their position within the image. The Gaussian network is used in a hierarchical structure that allows the system to learn basic features at the first level, and combine these features in later layers to produce higher level features. Only the last two layers are classification layers which know the classes represented by the input. The architecture is very similar to the Neocognitron described by Fukushima [4] (Figure 18).

The system consists of an input image, one or more feature detection modules, and a classifier. The classifier is just the last two layers of the Gaussian classification system described above. The feature detection module consists of a layer of Gaussian feature detectors followed by a layer that combines the different individual features in a small area on the feature detection layer. Each layer consists of one or more slabs. A slab consists of nodes which share the same mean vector and covariance matrix. In this manner, each slab can be thought of as a single feature detector, with each node in the slab looking for the same feature on a different area of the previous layer.

Figure 18 shows the overall architecture. It consists of an input layer, the visual field, one or more feature detection modules (only one is shown for clarity), and a classification module. The Gaussian feature detectors, labeled **F** in Figure 18, have a small receptive field on the previous layer. The receptive fields overlap, so that adjacent nodes have overlapping inputs. These nodes share means and covariances, so the implementation needs only one set of these for each slab. This layer then learns to recognize features in an unsupervised manner, with a separate slab needed for each feature. The next layer, **M**, has a similar architecture. Each node has a small receptive field, only in this case the node sees only the maximum value of the nodes in its receptive field. This means that the dimension of these nodes is equal to the number of slabs in the previous layer. This gives the network a measure of tolerance for deformations and slight translations. As in **M**, each node on a slab uses the same means and covariances. These two layers form a module, called the feature detector module. The final module, the classification module, consists of a layer of Gaussian classification nodes, **G**, whose input consists of the output of all the nodes in the final **M**

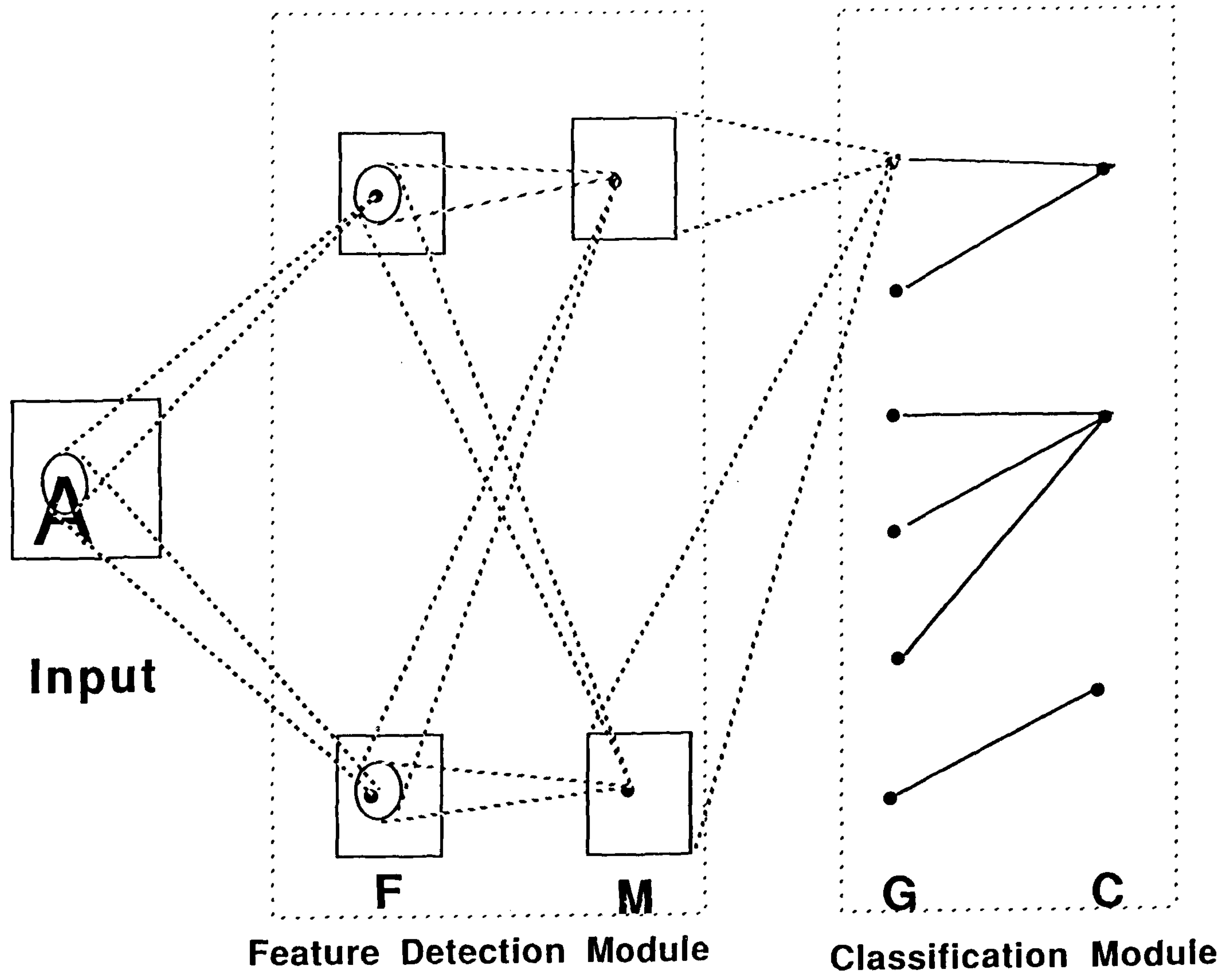

Figure 18. Image recognition architecture.

layer, with each node keeping its own mean vector and covariance matrix. These nodes are taught in a supervised manner, so they also have a class associated with them. The final layer, **C**, contains one node for each class (letter) that the network must recognize, and each node computes the weighted sum of the outputs from the nodes of its class on the **G** layer, as described in Section **IV**.

The system described makes strong use of the interpretation of Gaussian nodes as feature detectors. In addition, it uses an idea taken from biological visual systems, that of restricting the input, or receptive field, of the nodes to allow the system to discover local features. These low level features can aid in attention focusing. The areas of the visual plane that have a high level of activity amoung these feature detectors are the areas of likely interest. These low level features are combined in a hierarchical system to combine low level features into more general features.

This system has shown promise in recognizing characters under conditions of translation, noise and slight deformation, such as would be expected in handprinted characters. The problems of scale and rotation invariance still need to be addressed. A possible solution to the rotation invariance problem might be to use the low level features to focus attention, then implement a log-polar transform, which would give a measure of rotation invariance.

VII. Discussion

An adaptive network architecture has been described which draws upon conventional statistical methods for its theoretical groundings. Falling somewhere between the single Gaussian classifier and the kernel estimator, the system is concerned with developing a statistical representation of the input data. A two-faceted, one-pass learning algorithm, combining both connection-weight adaptation and transfer function adaptation, is developed from standard statistical moving mean and moving covariance calculations.

Several choices can be made in the learning algorithm described. On a high level, learning consists of adapting the means and/or the covariances. If the means are held fixed, the system is comparable to an adaptive kernel estimator, but it makes use of fewer nodes than the kernel estimator. If the means are allowed to vary, the Gaussian nodes tend to cluster around the modes of the distribution. This allows the system to seek out "features" in the data, and it becomes a classification system based on these features. The system can be taught in either a supervised or unsupervised manner, or these methods can be combined to allow the system to continue to adapt after the initial supervised learning. On a lower level, the constants **u** and **c**, which control updating and creation of nodes, can be varied. These constants control the smoothness of the distribution and the number of nodes used by the network. More work is needed to determine intelligent choices for these parameters for a given data set.

The architecture is a very flexible classification system which has been applied to a wide range of problems including robotic pattern recognition, vision and audition. It can be easily tailored to the problem, and the representation produced by the network is easily understood in statistical terms. The network can be designed within hardware constraints, such as the number of nodes available in an implementation, to produce a robust, application-dependent network. The combination of both supervised and unsupervised learning techniques yields a truly adaptive system which can continue to evolve after initial teaching.

Acknowledgements: The authors would like to thank Dr. Dennis Healy, Dr. Steve Roeder, and Susan Alderson for their help, support and encouragement.

The authors are with the Naval Ocean Systems Center, Architecture and Applied Research Branch, Code 421, San Diego, Ca. 92152 USA.

VIII. References

[1] Chan, Golub, & LeVeque, "Algorithms for computing the sample variance: analysis and recommendations", The American Statistician, Vol. 37, No. 3, pp. 242-247, 1983.

[2] E. F. Codd, *Cellular Automata*, Academic Press, 1968.

[3] H. Everett, G. Gilbreath, S. Alderson, D. J. Marchette & C. E. Priebe, "Intelligent Security Assessment for a Mobile Sentry Robot", Proc. Institute of Nuclear Materials Management, 1988.

[4] K. Fukushima & S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position", Pattern Recognition, Vol. 15, No. 6, pp. 455-469, 1982.

[5] S. Grossberg in *Pattern Recognition by Humans and Machines*, Vol 1, eds. Schwab & Nusbaum, Academic Press, 216-226, 1986.

[6] B. Kedem, "Spectral analysis and discrimination by zero-crossings", Proc. IEEE, Vol. 74, No. 11, pp. 1477-1493, 1986.

[7] F. P. Preparata & M. I. Shamos, *Computaional Geometry: An Introduction*, Springer-Verlag, 1985.

[8] C. E. Priebe, "Temporal Information Processing", Master's Thesis, San Diego State University, 1988.

[9] C. E. Priebe & D. J. Marchette, "Temporal Pattern Recognition: A network architecture for multi-sensor fusion", to appear in SPIE 1988 Cambridge Symposium on Advances in Intelligent Robotic Systems.

[10] L. R. Rabiner & R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Inc, 1978.

[11] D. E. Rumelhart & J. L. McClelland, *Parallel Distributed Processing*, Vol I, Bradford Books/MIT Press, 1986.

[12] G. A. F. Seber, *Multivariate Observations*, Wiley, 1984.

[13] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman anu Hill, 1986.

[14] D. W. Tank & J. J. Hopfield, "Neural computation by concentrating information in time", Proc. Natl. Acad. Sci. USA 84, pp. 1896-1900, 1987.