RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

IN-63
43065

Distributed Memory Approaches for Robotic Neural Controllers

P71

Charles C. Jorgensen
Research Institute for Advanced Computer Science
NASA Ames Research Center

August 1990

RIACS Technical Report 90.29

NASA Cooperative Agreement Number NCC 2-387

Distributed Memory Approaches for Robotic Neural Controllers

Charles C. Jorgensen
Research Institute for Advanced Computer Science
NASA Ames Research Center[1]

RIACS Technical Report 90.29

NASA Cooperative Agreement Number NCC 2-387

1. Work reported herein was supported by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA) with contributions from the Information Sciences Division of Ames Research Center and the Information Systems Division of Langley Research Center.

Abstract

This paper explores the suitability of two varieties of distributed memory neural networks as trainable controllers for a simulated robotics task. The task requires that two cameras observe an arbitrary target point in space. Coordinates of the target on the camera image planes are passed to a neural controller which must learn to solve the inverse kinematics of a manipulator with one revolute and two prismatic joints. Two new network designs are evaluated. The first called a radial basis sparse distributed memory or RBSDM, approximates functional mappings as sums of multivariate gaussians centered around previously learned patterns. New patterns are categorized using gaussian distances from learned examples with training patterns as stored as high dimensional input addresses in a sparse distributed memory architecture.

The second network types involved variations of Adaptive Vector Quantizers or Self Organizing Maps. In these networks random, N dimensional points are given local connectivities. They are then exposed to training patterns and readjust their locations based on a nearest neighbor rule. The winning point and its neighbors are dragged differentially toward the new pattern, adjusting to minimize the elastic global energy of the network. The result is a network that adaptively forms an interpolating n-dimensional surface over the density of the training sample set. A new learning rule is proposed called the proportional winner rule, which dramatically simplifies problems in learning rate and radius scheduling. A new network called an infolding net is presented which has advantages of a self organizing map with superior learning performance and potential for real time control.

Both approaches are evaluated based on their ability to interpolate manipulator joint coordinates for simulated arm movement while simultaneously performing stereo fusion of the camera data. Details of the benchmark task developed to compare the two models are contained in a companion report titled "Development of Sensor Based Kinematic Models for Neural Network Controller Training" (Jorgensen 1990). Comparisons are made with classical k-nearest neighbor pattern recognition techniques and a procedure for application testing with real hardware is described.[2]

2. Acknowledgment: I would like to thank the Sparse Distributed Memory Group at RIACS for their support and helpful discussions during this research. In particular I would like to thank Dr. Doug Danforth for his helpful suggestions regarding simplification of learning terms in a SOM, and Dr. Mike Raugh for providing the research environment which made this work possible.

Introduction

An issue of increasing importance for automated space operations involves the integration of multiple sensor inputs with robotic control. A number of difficult questions about controller design inevitably occur as the complexity of required tasks and the variability of the application environment increases. In certain situations (e.g. high degrees of freedom, dynamic environments, or non-linearities) traditional methods to formally specify controller behavior may not be computationally efficient with potential system time constraints and environmental variability. This has led to an interest in trainable forms of controllers.

One area offering considerable promise is artificial neuromorphic systems or 'neural network' methods. Research in this field has been increasing in recent years. In many cases, however, the evaluations of the methods have been confined to domains particularly well suited in scale and problem type, e.g. visual pattern recognition. More recently attention has been given to application of network methods to control[3]. Among the more vexing questions are whether network solutions can generalize to new situations, if they can capture functions with nonlinear discontinuities, and if they can integrate human knowledge and constraints. The present research was motivated by a desire to evaluate a particular kind of network in the context of realistic problems. These were problems which can be solved using current control techniques but represented a minimum capability for neural networks if they were to have hope for addressing issues of much greater difficulty NASA is confronting as part of the Space Exploration Initiative.

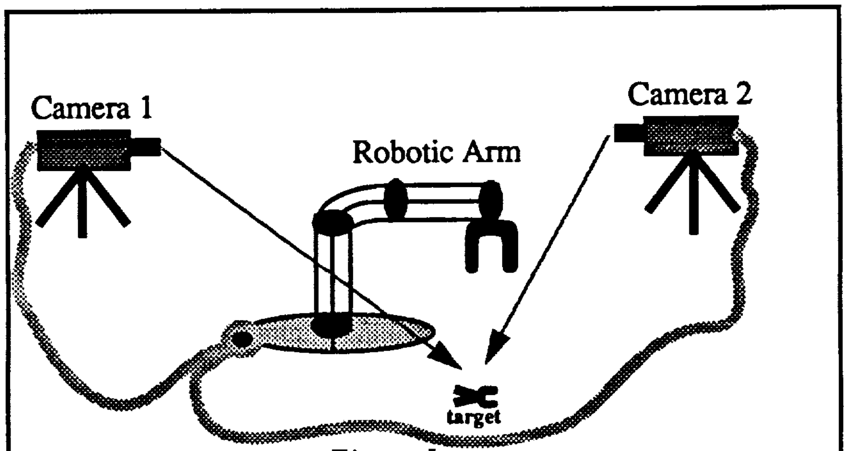


Figure 1.
Coordinated Movement

Figure 1 presents a typical situation which might occur in space-based assembly or autonomous exploration. Two or more cameras (or other sensors) are sited at variable positions. The task involves having the cameras fixate on a target and send location coordinates to a controller which uses the coordinates to calculate joint angles. Two mappings are required to complete this sequence. First, the relationships between camera

3. Miller, T.," Neural Robotics and Control", MIT Press, 1990.

projections must be unified so a pair of 2-D images correspond to a single 3-D point (stereo fusion). Second, inverse kinematics must be solved to map x,y,z coordinates to values in joint space at the correct position and orientation for the arm tip to touch the target.

Because of complex interactions between the number of degrees of freedom and the possible joint coordinates, this mapping does not necessarily have a unique inverse which means practically that there may be many different joint combinations which can reach the same point. In a companion volume to this report a benchmark is developed which can be used to produce data for evaluating alternative neural network controller concepts. The training information derived from this model consisted of a series of input /output pairs composed of four camera coordinates and three joint space angles. This data was used in the present paper to compare two different network approaches to control. Evaluation was based upon the networks joint angle predictions for new camera coordinates relative to exact angles derived from inverse kinematic equations of the arm and camera models.

Several questions motivated the investigation. First, earlier research[4] indicated that for aircraft landing control, methods which relied on nonlinear regression (i.e. back propagation) became very unstable when trained on data sets having discontinuities. Such discontinuities were produced by abrupt system changes such as transitions in flight control modes. Attempts to find a single smooth function accounting for the data resulted in oscillations similar to the Gibbs phenomena in Fourier series.

This difficulty and issues associated with the generation and adequacy of training sets, led to a search for alternative methods to perform control and ultimately to consideration of distributed memory architectures. These included CMAC, SDM, and PNN networks in which a control action for a particular system state is stored as an instance in a high dimensional space. For these types of networks, when a new pattern is presented, output values are generated based on similarity of the observed pattern to previous system experiences. Pattern similarity may be defined in a number of ways such as using a Bayesian classification (PNN)[5], nearest neighborhoods (CMAC)[6], or Hamming distances in a memory address space (SDM).[7]

Distributed memory models such as the CMAC network have certain design advantages in that they are simple to train and are not subject to sudden discontinuities if present in the function they are modelling. Their biggest disadvantage is that they can require a large amount of storage that increases as a function of the dimensionality of the problem. Further they may not interpolate depending upon their design but return only the

---

4. Jorgensen C. and Schley C., Development of a Neural Network Benchmark for Autolander Control," To appear in Neural Networks for Control, T. Miller E., MIT Press, 1990

5. Specht, D., "Probabilistic Neural Networks, "Neural Networks Journal, Vol. 3, Number 1,1990.

6. Albus, J.," Brains, Behavior, and Robotics," Byte Publications, 1981.

7. Kinerva, P, "Sparse Distributed Memory," MIT press, 1988.

nearest neighbor among previously stored addresses (as will be shown in this paper, variations on distributed memory models can compensate for some of these weaknesses).

Another category of networks called Self Organizing Maps (SOM) have a different strength. Because they are inherently adaptive, they can adjust to changing phenomena, interpolate using a predefined number of representative data points, and can be combined to create composite interacting process models.[8] Their biggest disadvantage is that they are very sensitive to learning parameters and their training schedules, must often be fine tuned, have been traditionally applied to two-dimensional mapping problems, and can be unstable depending upon neighborhood constraints.

A major motivator for the present research was handling data discontinuities, hence questions about control adequacy were tested by requiring distributed networks to integrate sensor data (multiple cameras) with manipulation (a simulated robotic arm). The test environment included discontinuities at the limits of the robot arm joint angles so we were able to test the ability of the current methods to deal with the type of conditions that had proved difficult for backpropagation networks.

Since the benchmark problem included multiple cooperating devices, we were also able to explore the potential of these networks to learn integrated system operations from observation of correct input /output behaviors. Finally, the task environment was exactly modeled by a simulation so it was possible to study how well the networks could interpolate new values, react to changing resolution, and different learning rules.

Camera Model

Before presenting the specific networks that were developed, a brief description of the equations underlying the learning problem will be presented. Given a vector of target coordinates, what we required was a function Θ such that:
$\Theta(x, y, z) \Rightarrow (x_1, y_1, x_2, y_2)$ where the subscripts refer to two camera viewpoints 1 and 2 respectively. Figure 2 illustrates the benchmark configuration used in this report..

In this figure, it can be seen that if we focus multiple cameras on a common target we need to take into account the angle of the camera plane relative to the axis of the scene, the field size onto which the image is projected, the distance between the centers of the cameras and the focal point locations. Good methods have been developed for handling the required transformations (rotation, skew, scale, and translation) using homogeneous coordinate systems[9]. For purposes of producing training data, we can simplify the situation considerably by letting each viewpoint lie on the Z-X axis of the target object's coordinate system[10]. For example, each camera can be placed so its projection plane P is

8. Martinetz, T.,Ritter, H., and Shulten, K. "Three Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm, IEEE Transaction on Neural Networks Vol. 1, No.1, March 1990.

9. Ballard, D. and Brown, C, "Computer Vision" Prentice Hall, 1982.

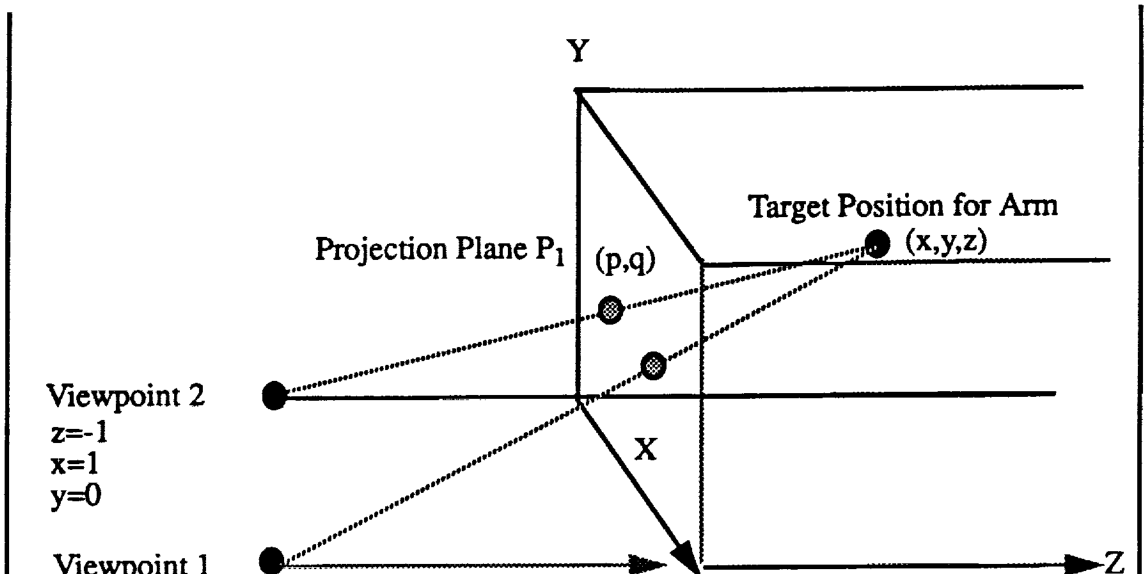


Figure 2.

Planar projection of point (x,y,z) on P_1

parallel to the x axis. If we let the focal plane also be an axis of one of the dimensions and place the cameras a distance 1 unit from each other at $y = 0$, then the relationship between the focal plane coordinates for (p_1,q_1) and the target point can be shown to be[11]:

$$\{p_1, q_1\} = \left\{\frac{x}{(1+z)}, \frac{y}{(1+z)}\right\}$$

where d is one half the distance between the center axis of the cameras and f is the focal length. Similarly for camera 2 and point (p_2,q_2):

$$\{p_2 q_2\} = \left\{\frac{1-x}{1+z}, \frac{y}{1+z}\right\}$$

These equations can be used to develop the camera image plane projections for arbitrary viewpoints but it is also necessary to derive the inverse of this transformation in order to have the set of relationships which must actually be learned by the neural network to produce x,y,z coordinates from camera coordinates.These are:

10. Longuet-Higgins, H.C., "A Computer Algorithm for Reconstructing a Scene from Two Projections" In Readings in computer vision, Morgan Kaufmann, 1987.

11. Jorgensen, C.C., "Development of a Robotic Benchmark Problem", RIACS Technical report, 1990.

$$z = \left(\frac{1}{p_2 + p_1}\right) - 1 \qquad x = \frac{p_1}{p_1 + p_2} \qquad y = \frac{q_1}{p_1 + p_2}$$

Robot Arm Model

In addition to specification of the camera, it was also necessary to generate a model of a robot arm. For the present paper, our need was for a three degree of freedom arm extending its links within a unit volume. There are a number of decisions when generating higher degree of freedom models. To clarify why the present set was chosen, some background is helpful.

A robot arm is simply a combination of links and joints formed together in a chain with one end fixed and one free. Each joint is driven by an actuator. The free end, also called an end effector, is moved along a path by sequentially activating the joints. Thus it is necessary to know the displacement of a joint at each point in time with respect to a fixed reference called the base frame. A path for the end effector is then defined in terms of the movement of this frame. The companion volume to this paper presents a detailed discussion of the assumptions that must be made when generating an arm model.

For our purposes, a simple but useful arm model can be developed by ignoring some of the parameters of full models. In particular, complexity can be dramatically reduced by omitting link twist, permitting a revolute joint only at the base, and two prismatic joints, one connecting the base to the first link, and one connecting the first link to the second. The result is a manipulator such as Figure 3.

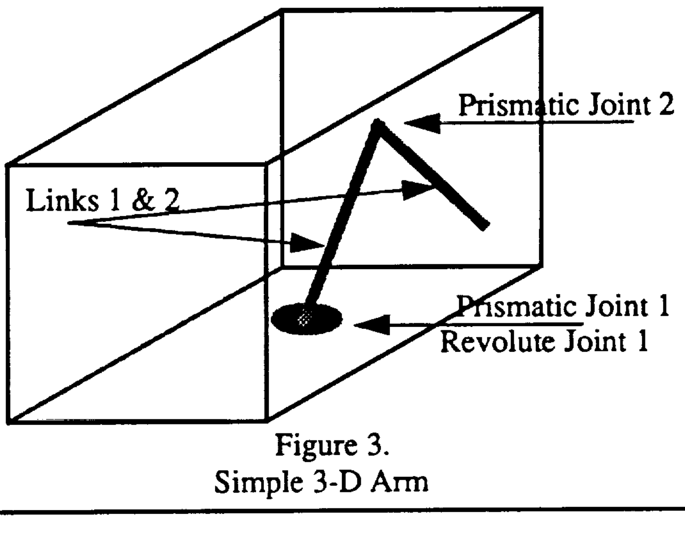


Figure 3.
Simple 3-D Arm

The forward and inverse kinematics of this manipulator can be derived as seen below.

The forward relationships are solved trigonometrically using the law of cosines through three intermediate terms based on theta one (the angle the arm is rotated on the x,y plane, theta 2 (the angle the first link makes with the x,y plane, and theta 3 (the angle link two makes with link one).

First we define the length d of a path between the center of the arm's base point and link 2 as:

$$d = \sqrt{(link_1)^2 + (link_2)^2 - 2link_1 link_2 \cos(\theta_3)}$$

and the angle theta tilde that this link makes with the y,x plane as:

$$\bar{\theta} = \theta_2 - \operatorname{acos}\left(\frac{(d^2 + (link_1)^2 - (link_2)^2)}{2dlink_1}\right)$$

We can then solve for the length of the radial projection r this segment makes on the y,x, plane as:

$$r = d\cos(\bar{\theta})$$

from which it follows that:

$$z = d\sin(\bar{\theta}) \qquad y = r\cos(\theta_1) \qquad x = r\sin(\theta_1)$$

In a similar fashion, we can derive a closed form inverse kinematic for this manipulator by defining d as:

$$d = \sqrt{y_2 + x_2 + z_2}$$

$$\theta_3 = \operatorname{acos}\left(\frac{((link_1)^2 + (link_2)^2 - d^2)}{2link_1 link_2}\right)$$

$$\theta_2 = \operatorname{asin}\left(\frac{z}{d}\right) + \operatorname{acos}\left(\frac{(d^2) + (link_1)^2 - (link_2)^2}{2dlink_1}\right)$$

$$\theta_1 = \operatorname{atan}\left(\frac{x}{y}\right)$$

Based on the above specifications, the required relationships are specified to generate training data for training neural network controllers. The training task can be divided into two parts shown in Figure 4. It is possible to use two neural networks, one for stereopsis

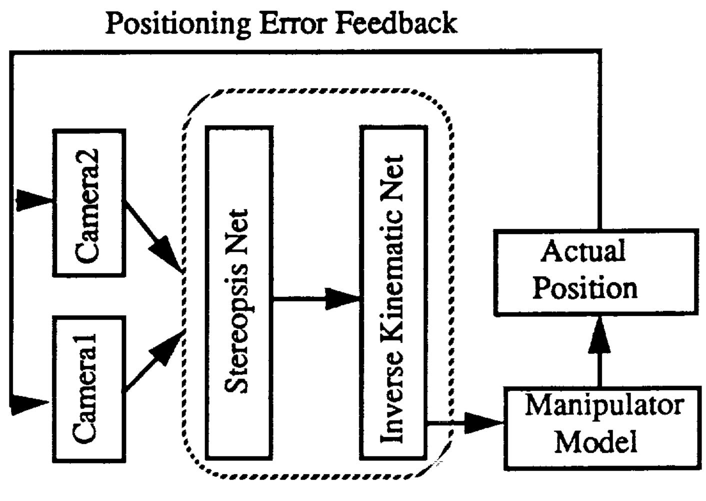


Figure 4 .
A Network Control Sequence

and one for control, or a single net taking camera input directly and outputing joint coordinates. In the present paper we used the latter approach since it can be shown that the two models can be made to link up automatically using one of the new networks that was developed (the infolding network).

Depending upon the type of neural network chosen (function fitting or distributed representation) and learning paradigm selected (such as supervised or unsupervised learning) very different training methods may be required. For example, in the case of distributed networks such as an SDM[12] or CMAC[13], a major consideration is the magnitude of samples required. With a self organizing map[14] a central training issue is whether the particular sample contains an underlying distribution similar to that of the process being sampled and what learning rate and neighborhood parameters should be used. In both cases however, testing involves presentations of part of the samples and evaluating the network's performance on the remaining points.

12. Kanerva, P. "Sparse Distributed Memory," MIT Press, Cambridge, MA, 1988.

13. Albus, J.S., "A Theory of Cerebellar Functions," Mathematical Biosciences, 10(1/2):25-61, 1971.

14. Kohonen, T. "Self Organization and Associative Memory," Spring Series in Information Sciences, Heidelberg, 1984.

Probabilistic Neural Network (PNN)

As mentioned, the initial motivator of this contract was to explore neural network alternatives that could handle potential discontinuities in a control process and still maintain a distributed representation which could be examined and seamlessly incorporate knowledge from many sources (e.g. human control inputs). Because neural networks using function interpolation were not likely to have such properties, examination of distributed memory techniques was undertaken. One kind, a feed-forward network called a Probabilistic Neural Net[15] showed potential for learning from observations to obtain the MAP value estimation in response to a new system input. Some early studies using the PNN as a method for landing a simulated Boeing 747-400 proved suggestive but were later found to scale poorly because the high dimensionality of the problem required an extremely large number of points to capture landing behaviors under variable wind conditions. Figure 5 shows how this network is connected. A series of input values fan

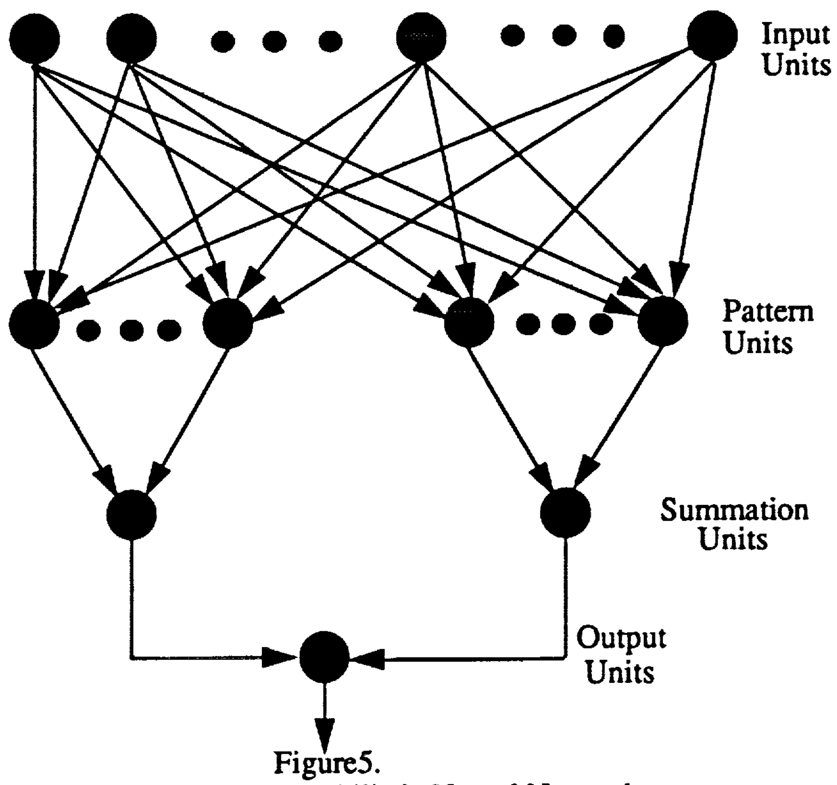


Figure5.
Structure of a Probabilistic Neural Network

into gaussian difference calculators called pattern units. Each input value is ranked in terms of its gaussian distance from the set of values centered at each pattern unit. Given m pattern units related to a particular output category (each having dimension p, values in vector psi, and an arbitrary input pattern x of dimension p), the output for all pattern units psi is shown below, where sigma is a term controlling the spread of the gaussian

15. Specht, D., Probabilistic Neural Networks for Classification, Mapping, or Associative Memory," Proceedings of the IJCNN, July 1988, Vol.1, pp.525-532.

$$\Xi = \frac{1}{(2\Pi)^{p/2}\sigma^{p}}\left[\frac{1}{m}\right]\left[\sum_{i=1}^{m} exp^{\left[\frac{-(\psi - x)^{t}(\psi - x)}{2\sigma^{2}}\right]}\right]$$

distribution. A selection is made such that the unit with the greatest response is the winner. Which output unit is selected depends upon both the sum of gaussians and a proportional weighting factor based on a ratio of prior probabilities and anticipated loss that is applied at the output units.

If the process is unknown, priors are frequently assumed to be equal. Loss functions are neglected simplifying the model. A sample of a PNN network which estimates a function value y based upon input values x was developed to test the method and is included in the appendix under the name CJPNNFIN.M. One of the nice features of this type of network was that it generated a smooth interpolation between near neighborhood points if a new data point fell somewhere off of the exact center of the multivariate gaussian distribution. How well the interpolation worked was determined by the value of sigma, which is found most easily by experimentation although theoretical derivation of its most effective value has been made.[16]

Another characteristic of the PNN net that made it attractive initially was that the degree of interpolation could be varied from a look-up table with very small sigma values to a sum of gaussians through larger sigma values. Generically, other gaussian calculation units have been studied under the label of radial-basis functions[17] in which the sigma values of the functions have been allowed to vary at each point to form interpolators with nonhomogeneous resolution across the function space.

Some of these nets applied a combination of back propagation learning and radial basis units to improve their mappings. A major problem with these nets however was how to determine the minimum number of units. For more complex problems they scale poorly because generally only a limited part of the possible training space is required to capture the function. Hence it seemed useful to develop models which are more memory efficient yet have the desirable interpolation characteristics of PNN or Radial Basis networks.

Sparse Distributed Memory (SDM)

One paradigm which designed for efficient use of memory is the Sparse Distributed Memory network of Kinerva[18]. SDM is a generalized random access memory suited for long (over 1000 bit) binary strings of data. Words serve as both addresses to and data for this memory. The key idea is similarity based addressing. That is, when accessed, the

---

16. Parzen, E. "On the Estimation of a Probability Density Function and Mode,: Ann. Math. Stat., Vol. 33, September 1962.

17. Poggio,T and Girosi, F. "A Theory of Networks for Approximation and Learning," MIT AI Memo No. 1140, CBIP Paper No. 31, July 1989.

18. Kinerva, P. "Sparse Distributed Memory," MIT Press, Cambridge, MA, 1988.

memory reads out not only the original write address but the contents of other addresses within a particular neighborhood (e.g. points within a Hamming distance of the address). As stated by Flynn, Kanerva, and Bhadkamkar[19]There are six concepts that are central to describing the behavior of an SDM. They are: writing to the memory, reading from the memory, the address pattern or cue, the data pattern or contents, the memory location or hard address, and the distance from the hard address.

For the present purposes, the most important characteristic of this memory is the way in which it deals with very large address requirements. Suppose for example that the robot control problem was to be learned using a small sigma look-up table PNN network. Each possible target point in three-dimensional space would have two different camera plane projections and three possible joint angles. Thus for each point there is a 7-dimensional address. If the minimum resolution for each coordinate was a coarse .01 with a range from 0 to 1, the number of possible addresses would be 100^7. Clearly, one would never be able to store enough training samples to completely describe this space.

SDM addresses the large memory problem by storing only a smaller (sparse) number of addresses in the larger space and upon retrieval from the memory forms an interpolation of the correct value based on a combination of addresses closest to that of the input pattern. An advantage of an SDM model is that a user can specify a predetermined number of physical memory addresses yet have the memory act as though it were representing a much larger dimensional problem, hence the name sparse distributed memory. As in the PNN network however, the mechanism to select the best representative addresses to use for an unknown process remains an active research problem.

In general, the SDM provides a way of looking at distributed memories that is well suited for hardware implementation and is likely to scale with problems that would be difficult for a standard PNN to handle. As a result, the first experiments for learning the control problem involved efforts to incorporate the best features of both Radial Basis Function Networks and SDM into a unifying structure. I have called such a hybrid a Radial Basis Sparse Distributed Memory Network or RBSDM.

19. Flynn, M.J., Kanerva, P., and Bhadkamkar, N. "Sparse Distributed Memory Principles of Operation", RIACS Tech. Report 89.53, December 1989.

Radial Basis SDM

There are two main ideas incorporated in the radial basis sparse distributed memory. First, learned patterns are represented by multivariate gaussian distributions with their centers located at the coordinates of previously stored points. Their shape is determined by the same sigma parameter defined above in the discussion of the PNN network. New patterns are compared to each previously stored point and assigned a similarity ranking based upon their summed gaussian distance from the pattern using all input dimensions. For the arm positioning problem the input dimensions were the four camera coordinates. The maximum response for a coordinate was obtained when the input data value exactly matched a stored point. As the point moved farther away, its value dropped off as a function of the shape of the distribution determined by sigma.

At this, the RBSDM diverges from a PNN in that it accesses the winning point like a sparse distributed memory where output values are calculated depending upon what type of reconstruction rule is chosen for the system. One simple rule is to pick the address with the maximum value. In this case the network functions as a nearest neighbor classifier based on a gaussian distance function. Another is to average values closest to the winning point in which case the network functions as a k-nearest neighbor classifier.

The use of the SDM distinctions in addresses and data, actually leads to a simplification of the PNN network. This is because the PNN is generally used as a Bayesian classifier producing MAP probabilities as a by-product of discrete categorization. A pure implementation requires breaking the output functions into seperate categories for each sub-range of each variable in the problem. For example if it was required that three joint angles be estimated for an input pattern of four camera coordinates, each angle would be treated as one category for each degree of joint movement. Therefore if the three joints were moved in a range of 90, 180, and 180 degrees, there would be 90x180x180 different category "bins" into which the classifier would associate input patterns. Clearly, the amount of data to get enough samples for each bin would be overwhelming. So a discrete categorical model would not be feasible.

On the other hand, a pure SDM architecture also traditionally represents addresses in terms of distinct bits. Its distances are defined using a Hamming metric between bit patterns (though other metrics have been explored[20]) and is usually formulated as a binary addressing device to facilitate hardware implementation. What it added to the development of the RBSDM however, was to make very clear a relationship between addresses for data, the content at the address, and the method used to access that location i.e. the reconstruction algorithm. If the input values are thought of as point on a continuum of addresses, then joint angles are merely contents found in that address's cell.

What this means practically is that the output layers of the PNN network are no longer required to maintain interpolation properties illustrated for that model. Data could also be

20. Jaeckel, L., "An Alternative Design for a Sparse Distributed Memory," RIACS Technical Report 89.28. (1989).

stored at address points using an SDM logic, however the RBSDM is different in that the metric applied to find the nearest address is a multivariate gaussian rather than a membership based upon a hamming radius within a discrete address space. This framework permits a method for dealing with continuous data, a graceful way to modify the address neighborhoods, and opens up the application of the adaptive point sampling schemes used by radial-basis models for SDM. It highlights a recurring need however, to modify SDM concepts to gracefully handle analog data.

Figure 6 presents a graphic of the RBSDM. Code implementing the model is included

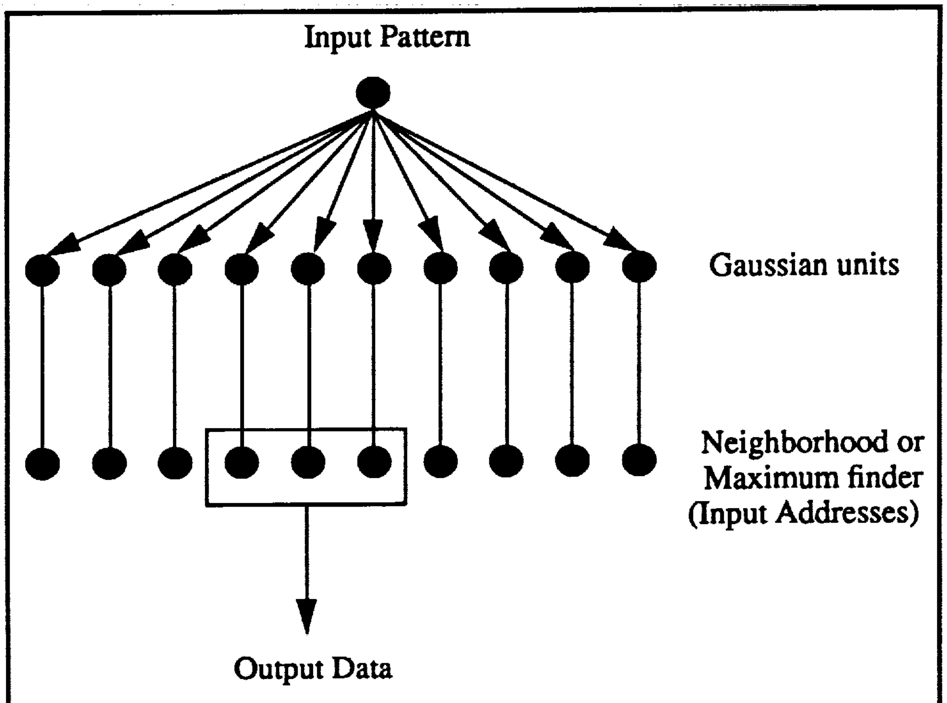


Figure 6.
Radial Basis Sparse Distributed Memory

in the appendix under the title of SDMPNN.M. The figure illustrates the significant difference from PNN which is the RBSDM recall procedure. An input pattern is presented to the network as though it were an address in a higher order space. This address is compared to other nearby addresses stored at points referenced by previously presented values. The distance between the addresses of those values and the input point is calculated as a sum of gaussians (one from each dimension of the input pattern). A neighborhood rule is applied which picks either the address having the largest sum as the winner or some neighborhood set of addresses. The output of the net is calculated as a function of the contents at those addresses. In the case of a nearest neighbor rule, the output will be a previously stored joint angle set. In the case of a neighborhood, the output value will be some function of the neighboring joint angles.

The RBSDM recall logic is also somewhat different than that of an SDM in that two parameters are used. The first is the value of sigma which controls how specific the network is in selecting relevant addresses. A second parameter called hrange was added which controls the limit within which the gaussian influence applies. Hrange is implemented by using a percentage of the maximum summed gaussian value. That is, its value reflects what percentage of the closest value a response most show before it is included in the neighborhood. Figure 7 shows the effects of changes in sigma and hrange.

As can be seen, the effective radius can be tuned by varying the cut-off value for hrange. If the gaussian is flat, the same hrange value produces a wider radius than if sigma is peaked. The more sigma approaches a look-up table, the less effect hrange has on the number of neighbors included. The RBSDM routine includes hrange and sigma in its call so that the user can directly influence the look-up and neighborhood characteristics of a given exercise "tuning" the network for optimum response to a given training set.

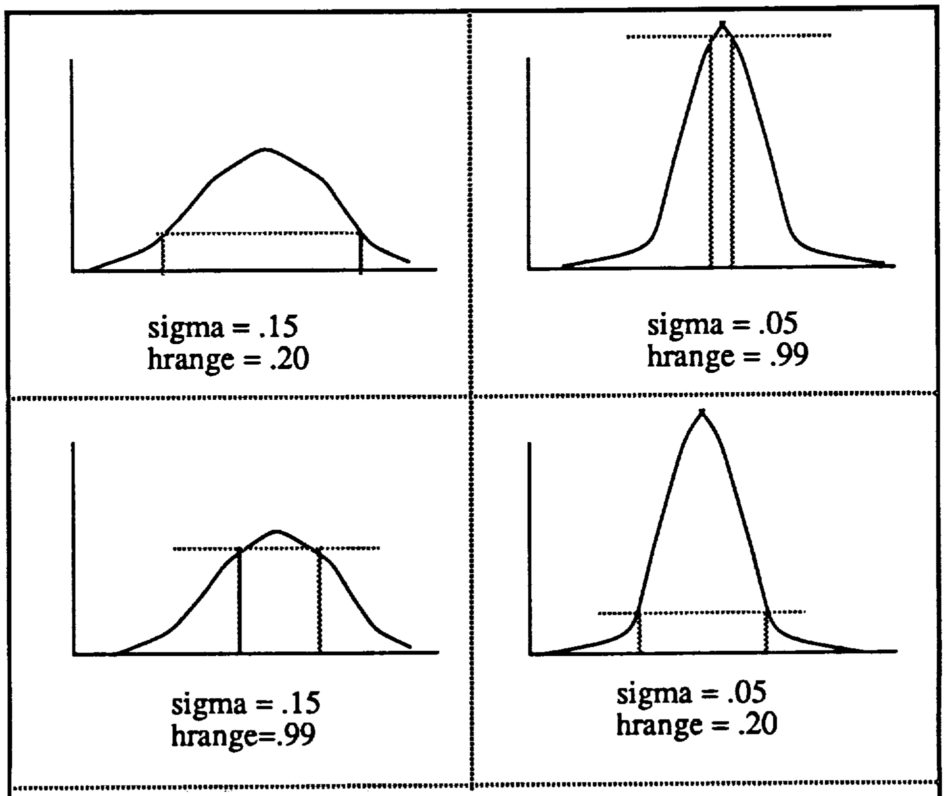


Figure 7.

Effects of Changes in Sigma and Hrange

RBSDM Performance

Table1 summarizes the performance of the RBSDM on the benchmark problem. The exact camera and arm models were used to generate a set of all locations reachable for the arm and visible for the cameras using the equations derived in the companion to this

Performance Statistics of RBSDM One Nearest Neighbor And Four Nearest Neighbor Recall Rules

Best RBSDM By Number of Points

	125	343	729	1000
Mean	.1207	.0758	.0579	.0515
Std. Dev.	.1453	.0638	.0605	.0381
Minimum	.0061	.0000	.0000	.0000
Maximum	.0986	.4539	.7746	.2328
Medinn	.0799	.0600	.0400	.0407

One Nearest Neighbor Rule on RBSDM

	125	343	729	1000
Mean	.0901	.0707	.0625	.0603
Std. Dev.	.0505	.0488	.0451	.0388
Minimum	.0061	.0061	.0061	.0061
Maximum	.2719	.2618	.3062	.2191
Median	.0866	.0587	.0550	.0585

Four Nearest Neighbor Rule on RBSDM

	125	343	729	1000
Mean	.0777	.0621	.0551	.0526
Std. Dev.	.0452	.0392	.0408	.0386
Minimum	.0000	.0054	.0000	.0000
Maximum	.2377	.2255	.2087	.1898
Median	.0684	.0519	.0445	.0428

Table 1

report[21]. One thousand points were selected at random from this set (total size 3800) to serve as potential training patterns hereafter referred to as the "trainset". The remaining points served as the pool of untrained points used to test the generalization of the network hereafter referred to as "testset". MATLAB procedures were written to randomly select varying size subsets of points from trainset to serve as previously learned addresses and reflect training sample effects for various amounts of training information.

For readers used to learning paradigms such as backpropagation, a comment about training distributed networks is in order here. Training for backpropagation requires repeated presentation of patterns during which weights are adjusted according to some error function. As a result learning is a difficult process. In contrast distributed memory networks deal with the interpretation of previously seen pattern values. Learning for these nets equals storage, and attention is on the selection and combination of representative points. The subtle aspects of the process center on reconstruction mechanisms during recall, which for backpropagation is predetermined by how the network was trained.

The test data in the table of the RBSDM was generated as follows. Previously untrained camera coordinates were presented to the network one at a time in random order. The maximum summed gaussian was calculated using a variety of values for sigma, hrange,and the number of estimating points. Joint coordinates were given to the arm model and an x,y,z end effector location was calculated for which the joint angles would have moved the arm. The positioning error between the network's arm position and the actual point locations were then calculated and stored as positioning errors. The process was repeated three to five hundred times for each tested set of parameter values and the mean positioning error, standard deviation, minimum error, maximum error and median errors were calculated.

Figure 8 shows plots of the mean positioning error as a function of the number of estimation points and the value of sigma with hrange held constant at .99.The first plotted values show the overall error performance of the network as a function of the number of points used to train. The mean error drops as the number of points is increased until leveling off at about .0515. Generally recall was better using nearest neighbor interpolation at recall rather than the gaussians. This was tested by fine-tuning the RBSDM to its best performance at a given number of points and comparing its mean error to the result of using only the raw data points and a one nearest neighbor or four nearest neighbor rule. As the number of points increased, this difference became negligible.

Table 2 and Figure 9. show how sensitive the network was to variations in sigma. As can be seen, mean error drops as sigma moves from .01 to .13 and gradually increases again after that value. This graph illustrates two points. First, it is relatively easy to empirically locate the most effective sigma value. Second, the network is forgiving over a fairly wide range of sigma values once the standard deviation has been increased to a degree that permits interpolation. In general, the best mean error and minimum standard deviation occurred at approximately sigma =.10, hrange .99 where the minimum mean

21. Jorgensen, C.C. "Development of a Sensor Coordinated Kinematic Model for Neural Network Controller Training," RIACS Technical Report 1990, April, 1990.

error was .0446 and standard deviation at .0339. Minimum error approached zero, maximum error approached .1649. Median error was .0334.

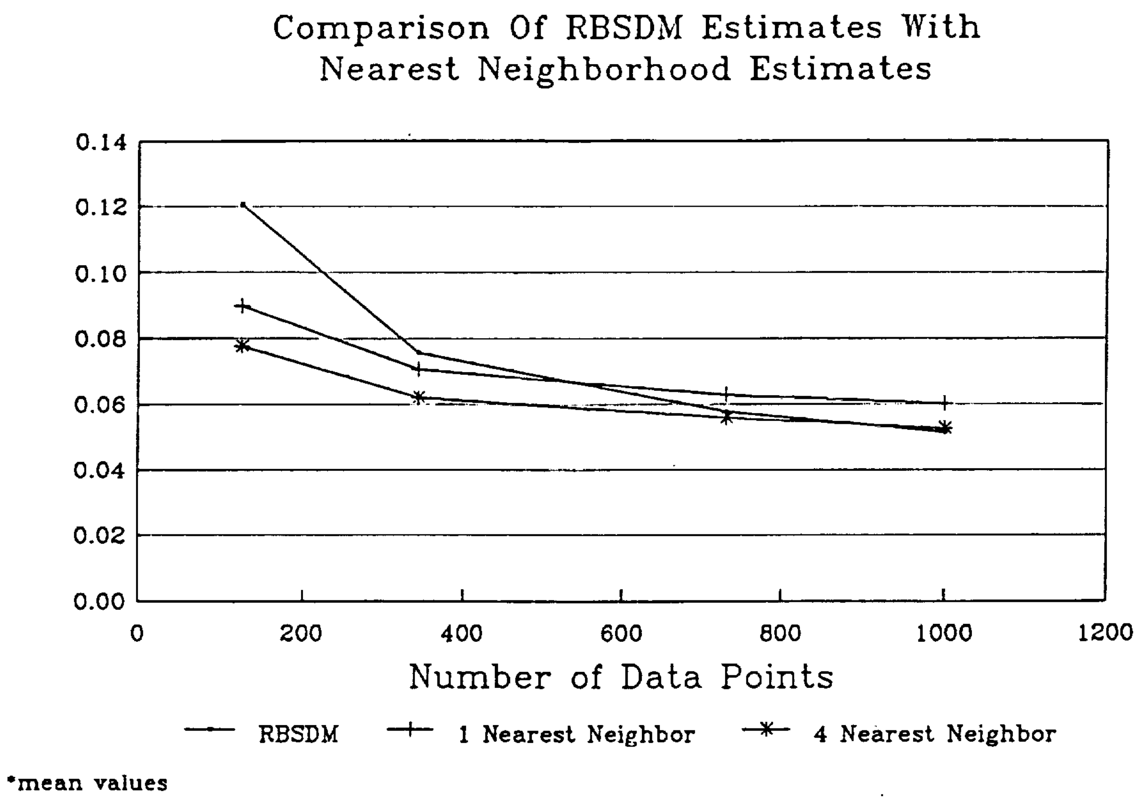
This performance can be assessed by comparing it to the average distance between points in the training set .2036 with standard deviation of .2704. Average interpoint distance in the testset was comparable at a mean of .1985 with standard deviation of .2651. As a check on the possibility that unique sampling effects might have occurred, a reversal test was performed, training with 1000 points taken from the testset and sampling 300 points from the training set. The results were comparable with mean error equalling .0408 and a lower standard deviation at .0294, minimum values at zero and maximum at .1591. Testing the recall of the RBSDM using previously stored points showed negligible error as expected.

Because the simple neighborhood rules performed so well, manipulation of the recall process was examined in more detail. One of the first things that was considered was to take advantage of the input pattern processing architecture to average estimated RBSDM output values based on their gaussian distance to the input pattern. That is, within a given neighborhood, the estimate of an output value was calculated based upon a weighted mean proportional to the gaussian distances from the input patterns. This averaging scheme did not prove effective.

As seen in Figure 8, a four nearest neighbor recall rule applied to the same addresses exceeded the performance of the RBSDM until the point size became very dense at which time little difference was observed. This implied that the RBSDM might do better if the gaussian dispersion was markedly increased. A test of how well the gaussians would smooth a very sparse grid with sigma at .2 and .1 and at alternative hrange values .02 and .5 showed very poor performance. The conclusion was that for limited numbers of data points, small nearest neighborhood interpolations using a standard euclidean distance proved superior. As the density of stored points increased, gaussian look-up provided equivalent and perhaps slightly superior estimates.

Although it had a desirable level of precision once the sigma values had been correctly determined, other properties of the RBSDM motivated the consideration of alternative distributed memory methods. Among its strengths were that storage of exemplars meant that previously encountered patterns would be correctly recalled with very high probability. However, when new patterns were presented, the net did not necessarily generalize well because the gaussian front end could not always form a smooth interpolating surface between previously presented points. This occurred when each point was given the same radius of influence (the same gaussian shape) hence the interpolations were more accurate where the training data was dense but suffered from disconnected "gaps" where the training data was sparse. Where these gaps existed, the net would return the nearest neighbor from among the set of previously trained points even if that point was some distance from the test pattern. Without overlaying some kind of averaging scheme upon recall, the network degrades into a gaussian look up-table.

Another difficulty dealt with a user's ability to control the adequacy of a training data set. It is generally not possible to specify in advance the correct number of training



	125	343	729	1000
RBSDM	.1207	.0758	.0570	.0515
1 NN	.0901	.0707	.0630	.0603
4 NN	.0777	.0621	.0559	.0526

Figure 8

RBSDM Performance As A Function Of Sigma

	.03	.06	.10	.12	.13	.15
Mean	.1071	.0561	.0446	.0451	.0460	.0429
SD	.1738	.0757	.0339	.0297	.0303	.0291
Max	.9414	.5198	.1649	.1214	.1484	.1628
Min	.0000	.0000	.0000	.0000	.0000	.0000
Med	.0514	.0365	.0334	.0400	.0367	.0367

*hrange = .99

	.20	.30	.40	.50	.60
Mean	.0398	0424	.0495	.0766	.0582
St.Dev.	.0258	.0347	.0371	.0934	.0382
Max.	.1113	.1887	.2463	.9057	.1776
Min.	.0000	.0000	.0060	.0059	.0000
Median	.0398	.0427	.0400	.0627	.0582

*hrange = .99

Table 2

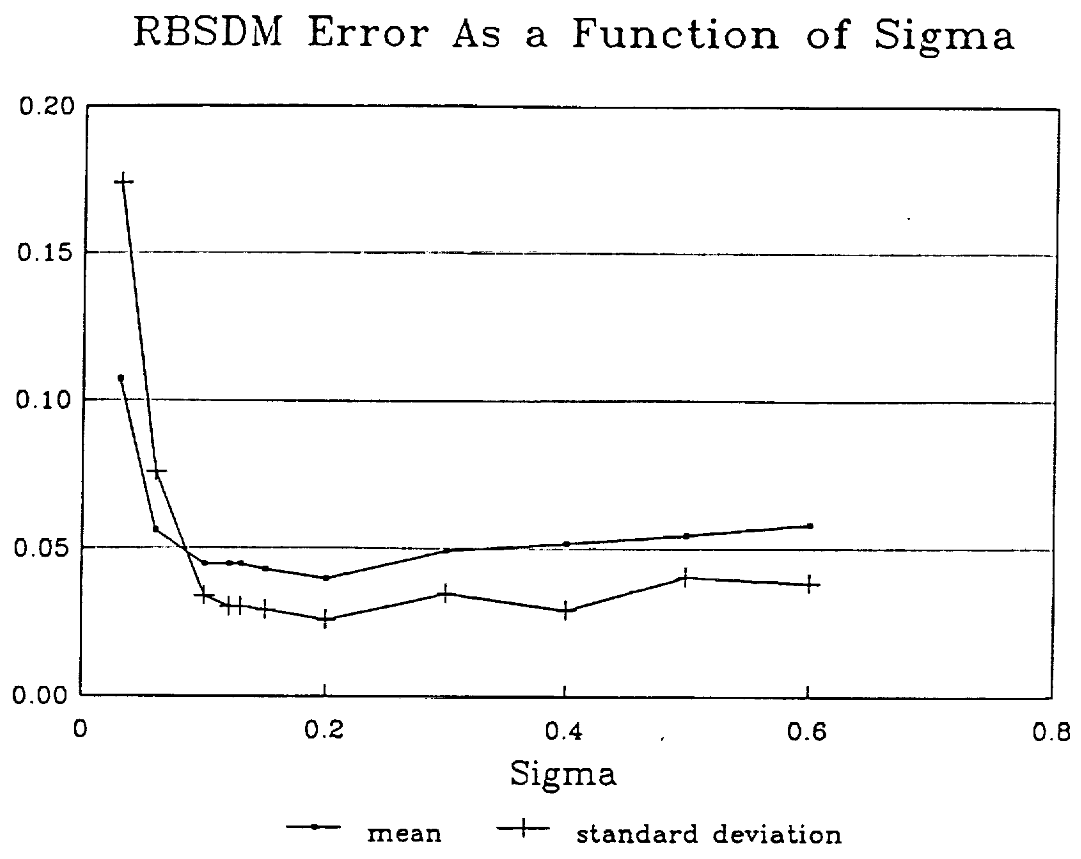

RBSDM Error As a Function of Sigma
0.20
0.15
0.10
0.05
0.00
0
0.2
0.4
0.6
0.8
Sigma
mean
standard deviation

Figure 9

samples and select correct sigma values if one were dealing with a changing real world process. In these situations, we would like to have networks adjust addresses for learned points to cover the space with a predetermined number of locations. This type of interpolation is handled by another network called a self-organizing map but is not well suited for the RBSDM or PNN memory architectures. The implication is that if the process is stable, RBSDM may provide an effective method for learning the control structure. On the other hand if the process varies in time, the method is not adaptive without introducing birth and death processes into the address space.

Storage addresses in a radial basis architecture are fixed by the training data. What varies are the number of stored points and the radius of the multivariate gaussian centered around the point. Although the gaussians can be expanded or reduced to interpolate better, point locations do not migrate over time. If the distribution of the training set does not closely match the underlying real-world problem, the network will not capture the central tendency of the set but only a particular instance of it.

If the underlying process is highly stable, this may not be a disadvantage. However, if the process changes or the sampled training distribution at a given time is unrepresentative, such an architecture may be inappropriate. A series of studies were therefore undertaken to examine the feasibility of using self organizing maps as an alternative storage mechanism for observed patterns. In addition to standard architectures, new variations were developed some of which proved extremely effective. Each of these is discussed in the following sections. We begin with a brief review of the key concepts of a self-organizing map.

Self Organizing Maps

The idea of a self-organizing map is generally attributed to Tueve Kohonen[22] although concepts of local neighborhoods and effects on nearest neighbors occur in a number of vision and pattern recognition techniques[23]. The central idea is to capture statistical characteristics of a training set by having the structure of a connected network reorganize in response to incoming patterns. The net is constructed in such a way that it has a fixed interconnection structure hence the shape of the network varies as it attempts to match the values of the incoming data set by moving current values of locations on the lattice. The net can perform higher to lower order mappings in cases where the input dimension is higher than the output dimension. It can also be used for a number of types of problems including pattern matching, speech recognition, and control.

The simpler versions of the algorithm work as follows. A number of points in the mapping space (usually coordinates in a 2-D plane) are selected at random. Associated with each point are other points chosen as neighbors and an output value such as a class name associated with each point's address. As the algorithm progresses, the locations of the points are moved toward the coordinate values of randomly presented training patterns. Formally this relationship is defined by two equations:

$$w_{t+1} = w_t + \alpha_t [i_t - w_t]$$

$$\alpha = f(t, r)$$

Where w is the weight vector for a particular unit, i is an input pattern at time t and

alpha is a learning rate term which is a function of time t and a radius of influence r. Alpha is generally a fairly simple proportional function of time and is controlled by a cooling schedule which gradually decreases the learning rate so the net eventually achieves stability. Similarly, convergence to a stable neighborhood size can be controlled by starting with radius which shrinks over time until a point is not perturbed by movements of its neighbors or using a fixed neighborhood and no variation in radius over time. The lack of a time varying radius results in a computation trade-off since many more trials may be required before the network reorganizes to a sufficient degree to capture the regularities in the training samples.

Within this deceptively simple framework, a large number of variations are possible which can result in marked differences in behavior. For example, if the radius is too large while the learning rate drops too fast, a nonoptimal map is generated that is overly influence by irrelevant patterns. On the other hand, if too small a radius is used and a slow learning rate scheduled, the net may not reorganize enough to minimize the global

22. Kohonen, T. "Self Organized Formation of Topologically Correct Feature Maps," Biological Cybernetics 44, 135-140, 1982.
23. Duda, R. O., and Hart, P. E.," Pattern Classification and Scene Analysis", John Wiley & Sons, 1973.

neighborhood constraints in the data. Research for this type of net usually focuses on finding particular schedules for alpha, and the nature of the neighborhood connections.

In the sections that follow, the tests of a SOM method for distributed control followed this procedure. After deriving a specific variation based on its theoretical or computational properties, a network was designed and implemented in MATLAB. The network was first tested for performance on a two-dimensional point set problem composed of a regular grid. If the network was able to capture the grid point ordering with 100 points, a full scale

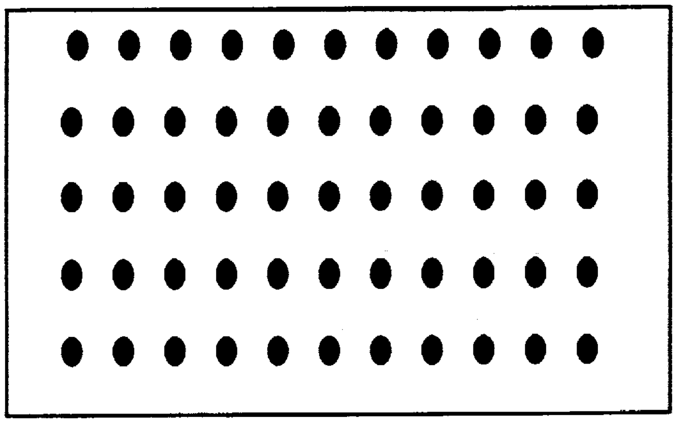


Figure 10.
2-D Grid Used To Test Models

version was coded for learning the 4-input 3-output, 7-dimensional problem produced by the arm problem. A number of variations of parameters were then tested. After learning, the network would be subjected random points corresponding to untrained but reachable arm positions. Typically each run would sample between 300 and a 1000 random points and plot the performance including values for mean error, standard deviation, minimum error for the set, maximum error for the set and median as was done for the RBSDM. At the end of the tests, comparative analysis were performed to select the best designs. These were later used as kernel architectures within which alternative learning rules were developed. Finally, comparisons were made between the SOM methods, the RBSDM, and traditional pattern recognition techniques (nearest neighbor and k-nearest neighbor classifiers). Source code for the most useful variations is included in the appendix to this report.

Quite a number of specialized networks were developed during the course of this research, some of which are sufficiently different that they might reasonably be classified as new types (e.g. the infolding net). These nets included higher dimensional self-organizing maps (3-D SOMs), new mapping procedures which studied the capture of higher order space by adjusting weights based upon a lower dimensional subset of that space, variations in learning rules (cooling schedules, coupled learning equations, and a new proportional winner rule), and variations in architectures (internal interactions within fixed neighborhoods, ranked nearest neighbor sets, stochastic neighborhoods, learning rate adjustments based on studies within the sparse distributed memory project (e.g. Danforth's polarity rule [24]), and generalization as a function of coarseness in approximation. Each of these variations will now be described.

Initial Studies

The first studies used two-dimensional SOM maps and the regular pattern grid to test performance. This was done for two reasons. First, most self-organizing maps in the literature have based their performance figures on 2-D planar problems. Second, the test data set could be given a regular form which simplified visual evaluation of the network as it learned. If the net was performing correctly, the points would start out as a tight, random cluster (although for some types of maps they are not required to do so) and gradually "unfold" until point locations matched that of the training set. A schematic of the evolution of such a net is shown below:

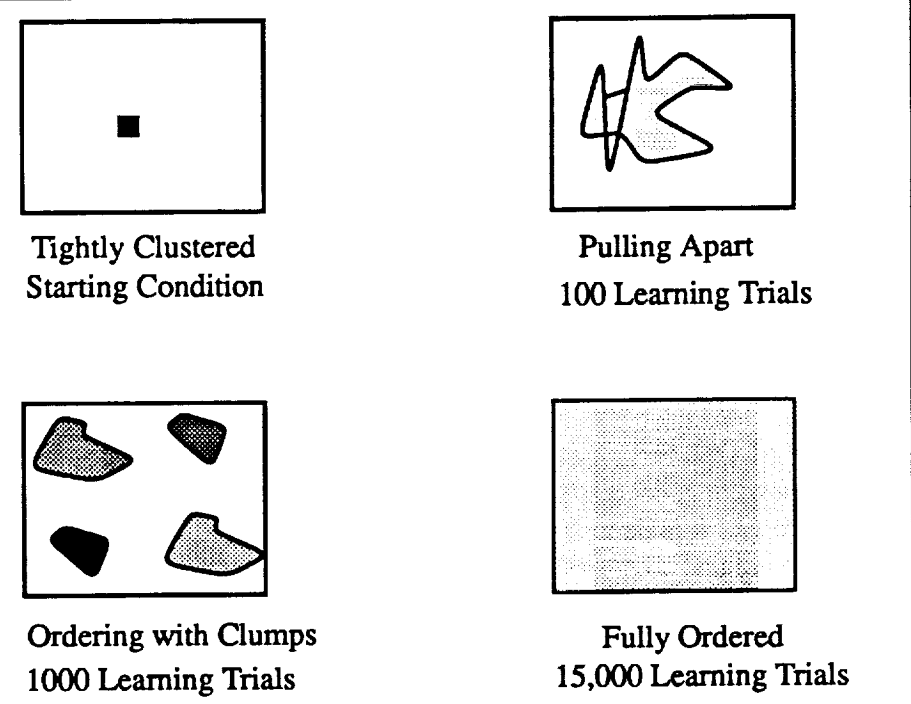


Figure 11.

Typical Evolution of a SOM Net During Training

What normally happens is that the points gradually pull away from the center cluster and begin to order themselves according to the neighborhood constraints and characteristics of the training data. This process continues until the point set topology approaches that of the generating function. Since the underlying distribution of the training points for this problem was random samples of a regular 2-D grid and had the same dimension as the SOM, the network conformed exactly to the generating function. If

24. Danforth, D.G. "An Empirical Investigation of Sparse Distributed Memory Using Discrete Speech Recognition" RIACS Technical Report 90.18, March 1990.

the training data had a higher dimension, for example 3-D, a 2-D net would attempt to twist into a representation of the projection of that space.

Because the projection of the network for a higher dimensional space could appear chaotic in lower dimensions, there are certain advantages in having the dimension of the SOM be matched to the underlying spatial dimensions of the problem. That is why a 3-D SOM was selected for the benchmark. By looking at the xy, xz, and yz projections of the map it became possible to determine whether the network was capturing the underlying movement constraints given to the robotic arm. For example since the arm was located in a corner and could only move 90 degrees in the xy plane (a quarter circle of reachable points), plotting that projection during learning showed whether the net was reorganizing to the correct spatial relationships for the arm. Figure 12 shows the projection of the entire set of reachable arm points on the xy, xz, and yz planes. Figure 13 shows a learned projection found by the best of the tested networks (the infolding network discussed later) given only a smaller set of random training samples. As can be seen, the network has captured the underlying structure of the reachable arm points very well indeed.

Such a visual, although useful to screen reasonable networks from poor ones, is not sufficiently precise for formal conclusions to be drawn. Consequently, after learning was completed, the nets were subjected to repeated performance tests through the presentation of randomly selected, previously untrained camera coordinates. The exact arm positions derived from the kinematic equations were compared to the positions reached by the simulated arm when given the joint coordinates output by the network. Because new points tested generalization and not look-up, the best performing of the network types were given evaluations for new and previously trained points.

Both the SOM variations and the RBSDM model permit different strategies for retrieving a controller estimate. The most straight-forward and the most efficient in this series of studies was nearest neighbor look-up. A simple modification permitted k nearest neighbor averaging and was used when there was reason to believe that interpolation performance might be superior. For the most part this modification did not provide a significant improvement when there was a very large number of representational points, although evidence for improvement did occur when the number of estimating points was reduced to less than 10 per cent of the 1000 points used as the largest number of points.

Contracting SOM

In this model weight learning was modified so that

$$w_{t+1} = w_t + \alpha\psi(i_t - w_t)$$

where w is a weight vector, alpha is a proportion between zero and one, and psi a factor effecting the maximum amount of change permitted per trial. Also required was a value for r less than or equal to .5 times the maximum range of values along the x,y, and z dimensions. In practice, r was found to work reasonably well at a value somewhere around three times the expected interpoint distance in the problem space. Since this would not

Axis Projection of Original Data Sets

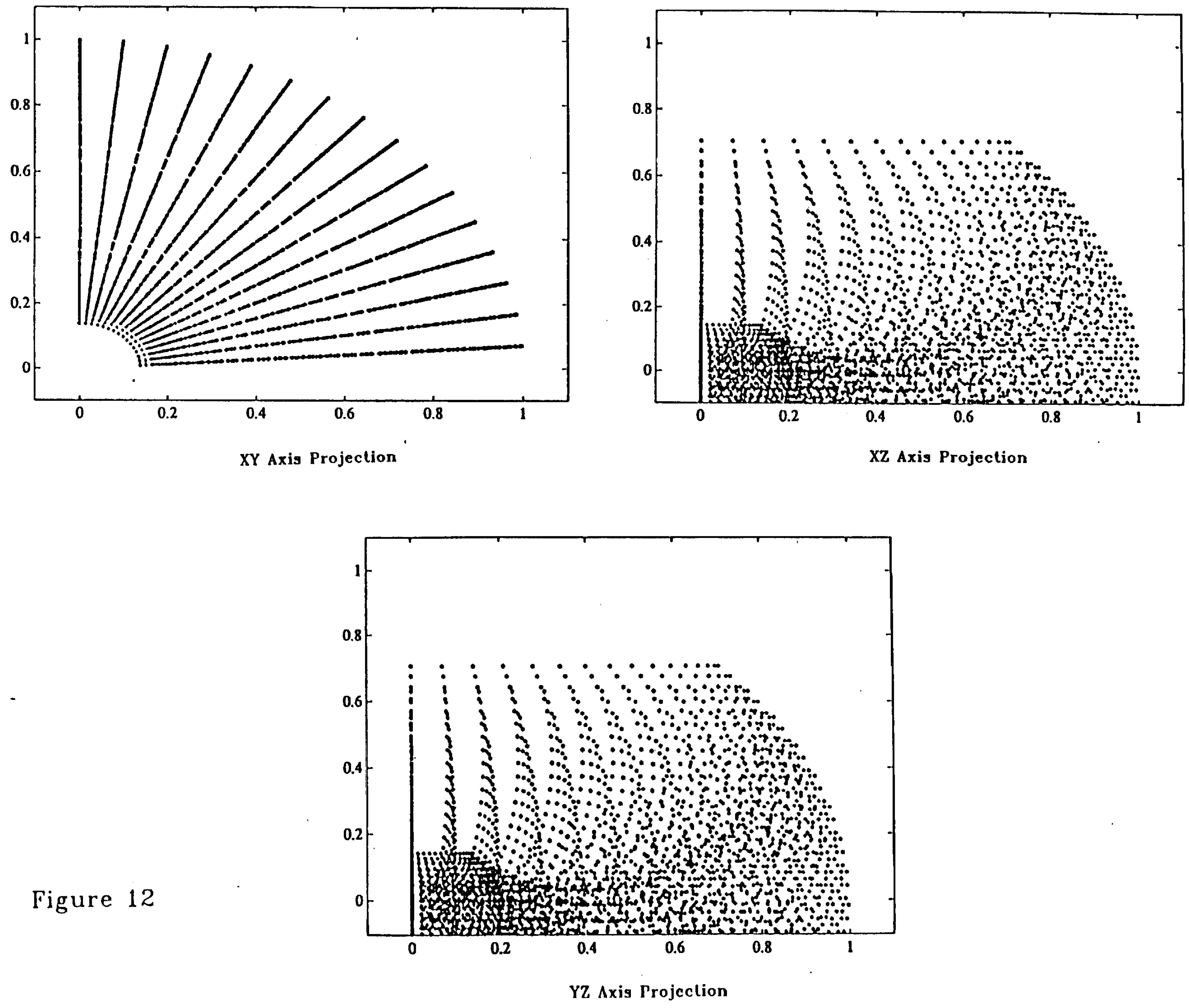


Figure 12

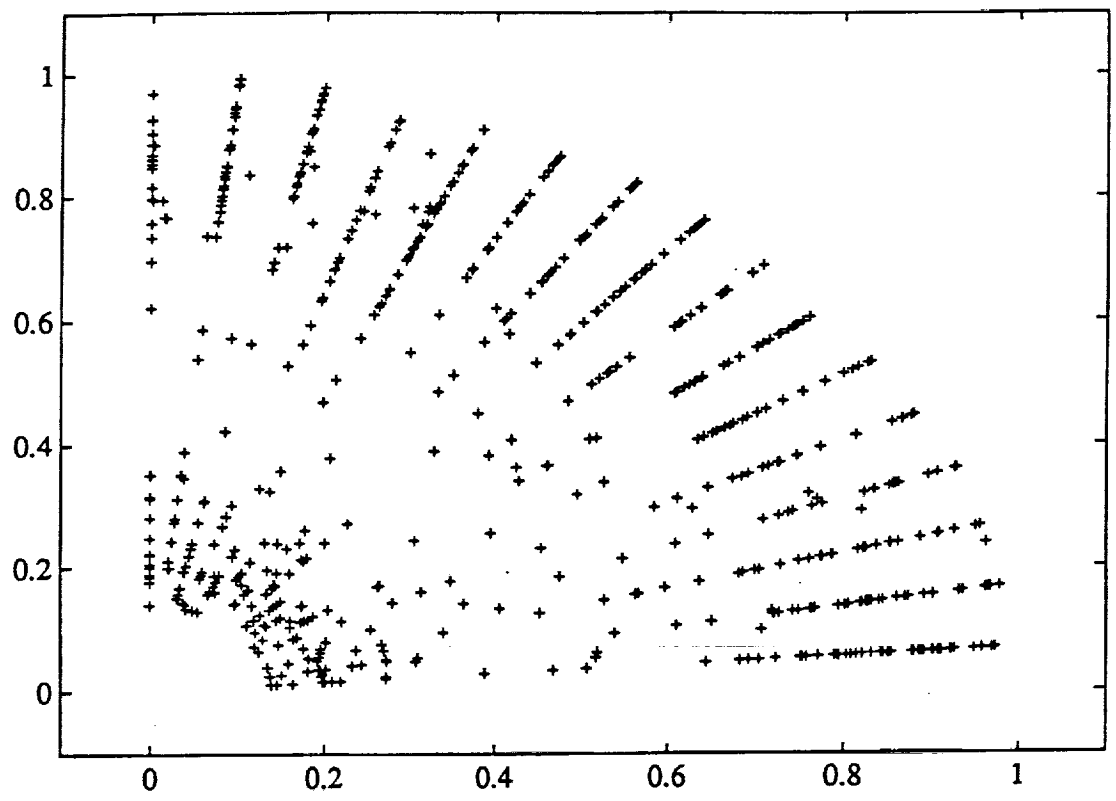


Figure 13

normally be known a priori, repeated adjustments on the rates of learning decrement, number of learning trials and the ratios between alpha and r values were usually required for each data set. The code implementing this model for the 2-D grid learning problem is presented in Appendix1 as SOM2DDEMO.M and SOM2DCODE.M. In this listing and the ones that follow it, there are usually two MATLAB M files. The first codes a user interface for changing training sets, selecting critical parameters, and choosing the number of learning trials. The second without the prefix 'test' or 't' includes the kernel of the calculations.

Initially it was speculated that psi should be defined so as to facilitate clustering of neighbors around active data points to lead to better generalization with untrained points. Thus points within the neighborhood radius were moved using a factor that minimized the migration of points already close to the input and maximized change for legitimate neighborhood points that were farther away. This strategy led to the title of a "Contracting" SOM.The learning proportion was to increase to a maximum adjustment of one hundred percent of the learning rate for distant neighbors and zero for close points. The logic was that although we might not know the correct starting locations for points in an unknown space we could sweep the space clear in areas where there were no active addresses and thus contract to a limited number of highly useful points. A learning rule which implements this logic is:

$$\psi = \left(1 - \left(\frac{min[i-w]}{max[i-w]}\right)\right)$$

where psi is the learning proportion, i an input training vector, and w a stored pattern vector. The graph below shows how this contraction rule worked for sets of 100 to 400 points. Unfortunately, the effect of the rule was to produce a 'dark star', pulling neighbors

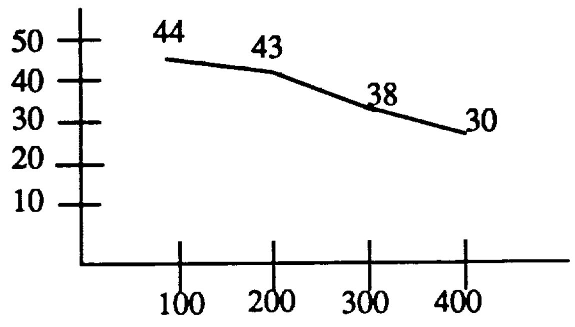


Figure 14.
Percentage of Points Absorbed by Attracting Adjustment Rule

into the same attractors, stacking too many of the points on top of each other. The network did begin to capture the grid structure but was wasteful of points and calculation time. Because of extreme sensitivity to the balance between alpha and the radius values, the network had to be hand-tuned for the correct learning and radius shrinking schedules. It was deemed unacceptable for applying to unknown learning environments.

It is interesting that depending upon learning schedules and their relation to the rates of radius contraction, the network could be made to behave as an attractor or repulsar, sort of a neural network version a dark star or an exploding universe. The latter would occur when points were pulled so hard they moved past the training point, took on negative values and began to push rather than pull in response to a neighborhood change.

After a number of experiments, it became clear that two problems needed to be resolved for the self-organizing map to become an effective control tool. First, the neighborhood relations would have to be balanced against the radius in some systematic and ideally, autonomous fashion. Second, the proportional learning rule favoring remote neighbors would have to be modified. A number of variations were explored including learning schedules that created maximum initial point spreading by keeping the learning radius large, then rapidly reduced the neighborhood radius while slowly decreasing the learning proportion. Other experiments looked at pulling all points only as much as the nearest neighbor, thus adjusting neighborhoods in homogeneous clusters. Different mixes of within-neighborhood relationships were explored such as attracting centers and repelling neighbors (the latter based on ideas drawn from the Mexican hat distribution useful for lateral inhibition networks in the retina). On the whole, none of these modifications were effective, and some, such as the lateral inhibition concept showed the expanding universe behavior which plagued the contracting nets. Thus it was decided to try a different tact. The first effort in this regard attempted to automate the balance between learning rates and radius through coupled differential equations.

Coupled Differential SOM

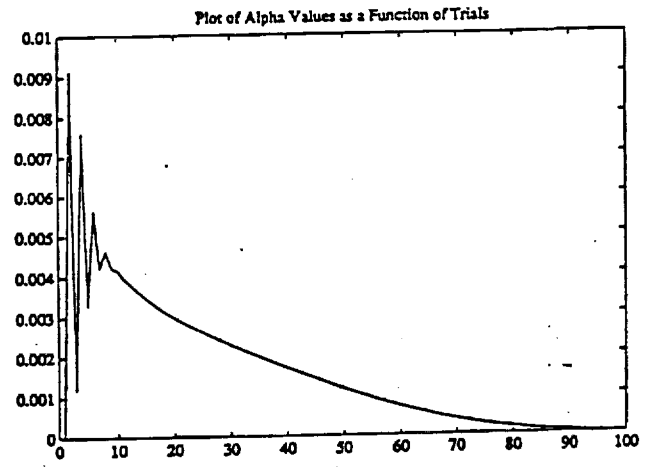
Some recent research on SOM nets has attempted to automate trade-offs between radius of influence and learning rates[25]. I explored these methods, but they were found to be unsatisfactory for the robotic control problem. Basically, the approach was to connect radius and learning rates through coupled differential equations such that if dt were the minimum distance between an input vector and all of its connecting neighbors, gamma the current minimum distance divided by the starting distance, and sigma the current radius divided by the starting radius, then the set of coupled equations was defined as:

$$\frac{dR(t)}{dt} = (-\sigma^{\gamma})\, e^{-(1-\alpha)}\, e^{-1/\sigma}\left(e^{-(1-\gamma)} - \sigma^2 e^{-\sigma^2}\right)$$

$$\frac{d\alpha(t)}{dt} = \sigma^2 \times \left[e^{-\sigma} e^{\frac{-1}{\gamma}} \frac{\tanh[1+\alpha\gamma]}{R^2(0)} - \alpha e^{-\alpha} \right] - \alpha$$

25. Hodges, R. And Wu, Chwan-Hwa, "A Method to Establish An Autonomous Self-Organizing Feature Map," Proceedings of the IJCNN-90, Washington D.C., Jan15-19,1990.

Performance of Learning Ratio and Radius As A Function of Trials For Coupled Differential Equations



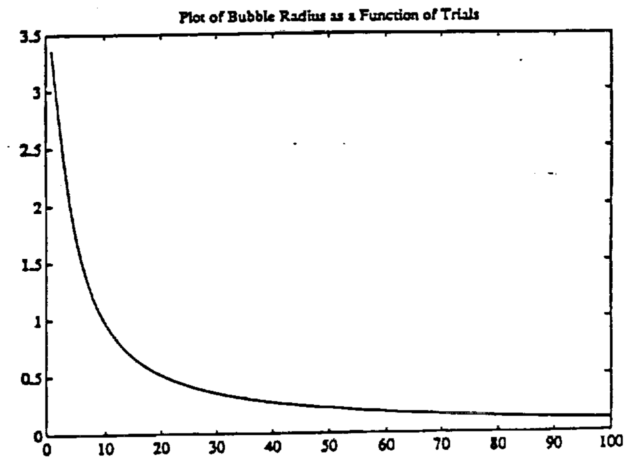


Figure 15

These equations were merged into the SOM written for the 2-D uniform grid and tested with varying initial rates of alpha and radius. The effect these equations had on alpha is shown in Figure 15. Alpha starts near zero and increases rapidly in an oscillating manner so as to "shake" the network's connection weights and in theory, produce a more uniform distribution of the weights within a certain neighborhood. Alpha increases until a balance point is achieved which is a function of the current value of alpha, the radius and the current closest neighbor. Once it has reached a maximum, it began to decay in a near exponential manner. Thus, in theory, leading to a rapid and balanced learning behavior.

This technique proved far too sensitive regarding the selection of parameters. Although it did produce a rapid decrease in the learning parameter alpha, it failed to provide an adequately stable map to capture the relationships necessary to learn the robotic problem. In addition to its complexity and difficulty in set- up, the technique had a high overhead in computational time.

Attempts to balance the rate of learning reduction through adjustments to the equation parameters proved extremely difficult and the method was finally abandoned as requiring too much fine tuning for the performance level obtained. Simpler designs using more learning trials and slower learning rates proved for the most part to be much easier to implement, require less computational load, and were more predictable. Code for a network implementing the coupled differential learning rule is found in the appendix under the titles of "TESTSOMRATES" and "SOFMRATES".

Becoming suspicious of the practical usefulness of a separate learning rate and radius approach, a new simpler rule was developed to replace alpha, r and psi with a single proportional term. The coupled equations did illustrate, however, that complex relationships may exist between neighborhoods, the problem space, and the particular data set used. Thus one important need was the development of a method for meeting the neighborhood requirements, while still preserving application advantages which a SOM theoretically provides. Most important among these was the potential for a more uniform interpolation to new data points while still maintaining control over coarseness of the learning grid (i.e. in this case the number of points in the SOM net that were selected initially to learn the training set). The SOM is also capable in theory of adaptive learning and hence a control system using such an architecture could modify itself over long periods of time and changing data distributions without fully discarding the desirable features of a distributed memory representation approach that led to the current research effort, namely resistance to sudden transitions in local behavior.

Proportional Winner Rule

After a number of experiments to find an alternative to coupled radius and learning rates, a simple, effective rule was found which I have called the proportional winner rule. It is defined as follows:

$$w_{(\xi t+1)} = w_{(\xi,t)} + \left[\frac{\|i_t - w_c\|}{\|i_t - w_{(\xi,t)}\|}\right][i_t - w_{(\xi,t)}] \qquad \forall (w_c, w_\xi \in W_r)$$

That is, a weight xi on trial t + 1 is adjusted based on its current value plus a proportion of that weight using differences between input, the closest weight at time t,and weight xi. The proportion is a ratio of distance between the input pattern and the closest weight in the neighborhood (w sub c) divided by the distance of weight xi from the input. W subscript r is the set of points within radius r of input i at time t ,c is the index of the nearest member of the neighborhood, and xi is the neighbor index.

The structure of this rule is interesting in that there is no longer an externally scheduled learning rate or radius. The winning point is moved directly to the coordinates of the input meaning that for at least the next trial, the network will function as a look up table for that point. This occurs because the minimum point distance divided by the current point distance is one if they are the same and proportional otherwise. The farther away a point is from the winner, the less it is effected, so the neighborhood influence on distant points is negligible. If a point already exists at the location, the distance is zero and no change in the network takes place. A nice property of the rule is that if no points are close, network points are effected to a greater degree. Thus the rule provides a powerful adaptive behavior but is extremely simple to implement. The ability to dispense with alpha, r, and learning proportions greatly simplifies the net and yet the power of the organizational properties remain intact.

Initial tests using the rule dramatically outperformed the approaches using automatic radius and cooling schedules. Table 3 and Figures 16a and 16b summarize the performance of a 3-D SOM using this rule on the robotic problem. Plots are presented for network sizes of 125, 343, 729, and 1000 points. Nearest neighborhood sizes at recall were varied from 1 to 20. The network was trained using a learning neighborhood of four points. It should be kept in mind during the remainder of the paper that there are two kinds of neighborhoods discussed. The first is the neighborhood used to organize the points during learning of the underlying distribution. The second is the number of points used to estimate a new value when the network is accessed during recall. They do not have to be the same size. Generally, a small learning neighborhood (about four) worked most efficiently. Unless otherwise noted, the plotted results using this size neighborhood during learning. The graphs reflect recall generalization for new points. Consequently, the neighborhoods shown on the plots are the size of the recall neighborhood used on a learned net of a set size. Overall, the plots show the best recall performance occurred using a local neighborhood of about four points. Both mean error and standard deviation were lowest. Other tests not shown indicated this result held constant even if the learning neighborhood was smaller or larger.As would be expected, generalization performance

Regular SOM Using Four Neighbors, And Proportional Winner Rule Modified By Index Of Neighborhood

(125 Points)

	1	3	4	6	20
Mean	.1109	.0992	.1061	.1049	.1449
St.Dev.	.0619	.0572	.0631	.0592	.0810
Max.	.2994	.3019	.3027	.3067	.4374
Min.	.0040	.0077	.0140	.0101	.0138
Median	.1005	.0873	.0922	.0935	.1301

(343 Points)

	1	3	4	6	20
Mean	.0816	.0724	.0703	.0743	.0849
St.Dev.	.0508	.0415	.0438	.0442	.0504
Max.	.2431	.2116	.2042	.2037	.2771
Min.	.0061	.0060	.0011	.0025	.0113
Median	.0707	.0685	.0589	.0713	.0716

(729 Points)

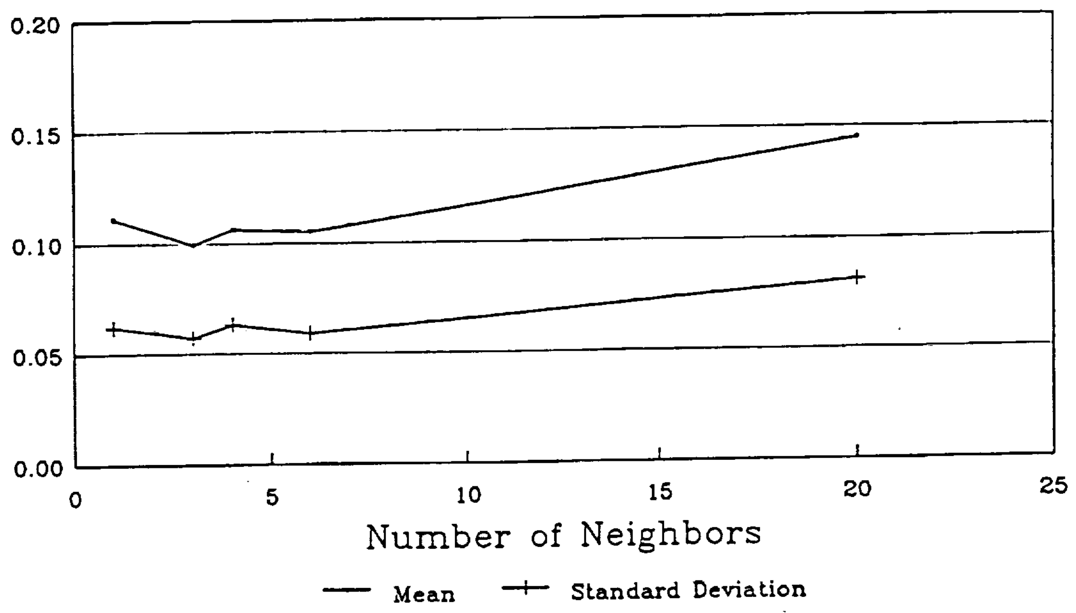
	1	3	4	6	20
Mean	.0697	.0587	.0667	.0655	.0722
St.Dev.	.0515	.0396	.0497	.0496	.0475
Max.	.2604	.2521	.2568	.2523	.2428
Min.	.0040	.0024	.0048	.0012	.0031
Median	.0560	.0513	.0533	.0516	.0596

(1000 Points)

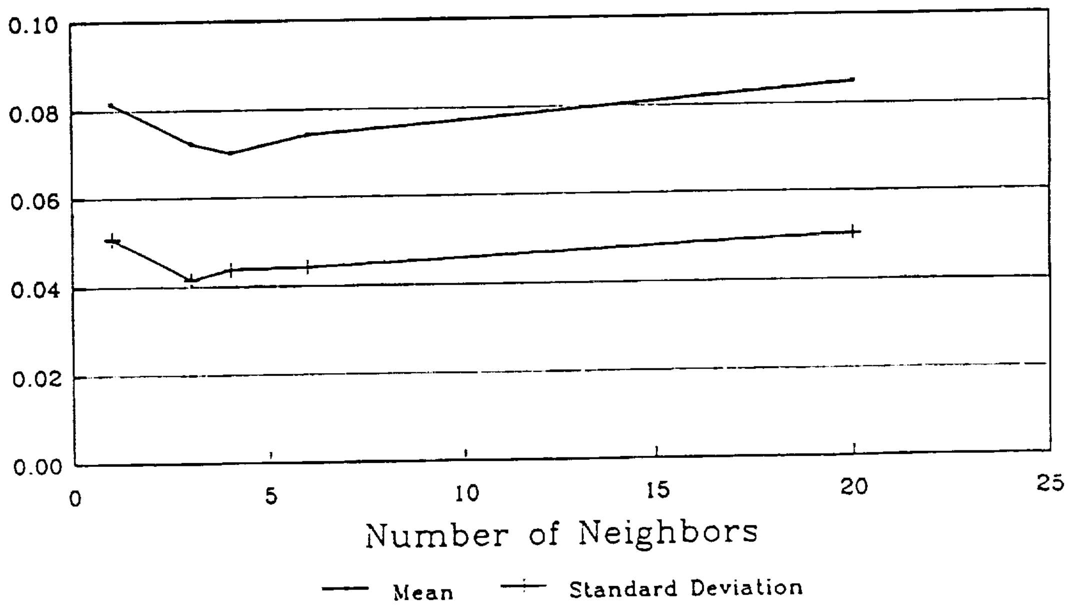
	1	3	4	6	20	30
Mean	.0616	.0576	.0609	.0599	.0592	.0656
SD	.0442	.0416	.0427	.0425	.0359	.0416
Max	.2604	.2450	.2740	.2519	.2249	.2349
Min	.0057	.0017	.0033	.0048	.0024	.0049
Med	.0490	.0473	.0609	.0495	.0510	.0543

Table 3

Regular SOM Using Four Neighbors, And Proportional Winner Rule Modified By Index Of Neighborhood



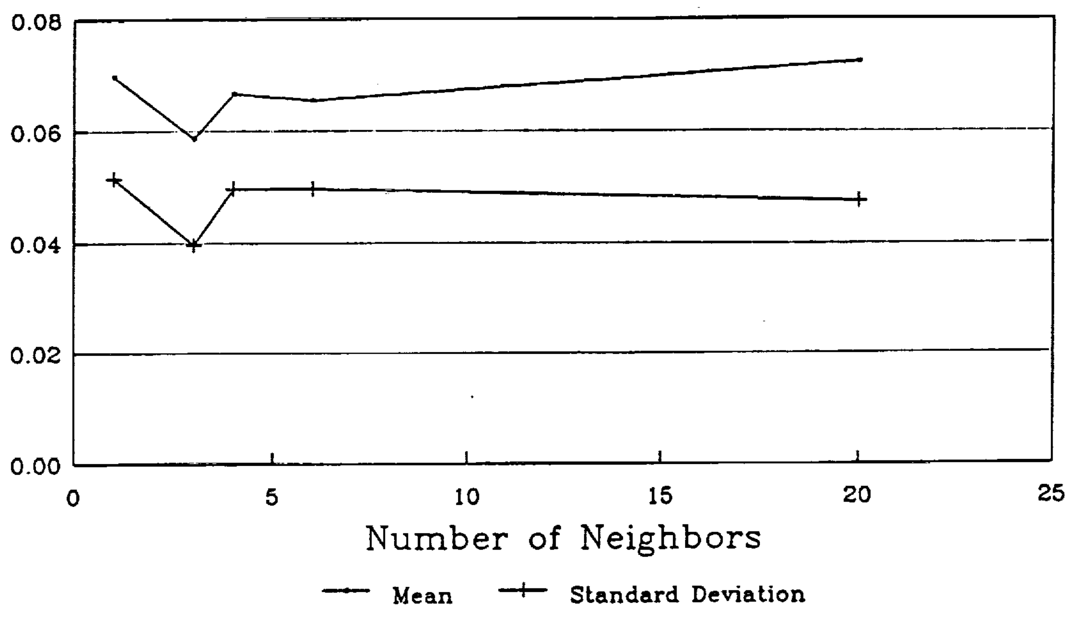
*125 data points



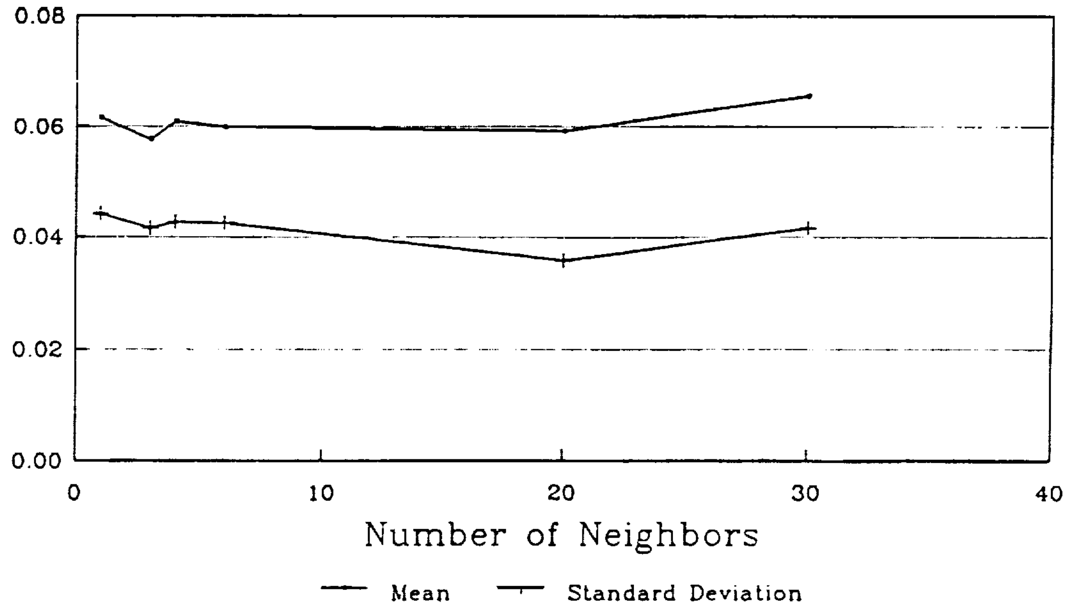
*343 data points

Figure 16a

Regular SOM Using Four Neighbors, And Proportional Winner Rule Modified By Index Of Neighborhood



*729 data points



*1000 data points

Figure 16b

increased as the number of points increased. The important point, however, is that performance for smaller number of points was better than that of the RBSDM.

One of the interesting aspects of this test was the metric chosen to define network neighbor proximity. Because there were four input dimensions (the planar camera coordinates) and three output dimensions (the robot arm joint angles required to reach the point represented by the planar coordinates) it is possible to use a seven-dimensional distance measure to determine how to adjust the weights relative to an x,y,z point. However, one of the potentials in using SOM maps was the possibility of having multiple controller networks learn autonomously and still connect smoothly to each other. Ideally, we would like to have one network control the relationship between points in the three-dimensional space and the sensor models. We would like another net to adaptively learn the inverse kinematic relationship between a point in space and the robot joint coordinate required to reach that point. This is because both processes might change with time at different rates (wear on the robot arm or distortion on lenses or sensing systems). If we required that all dimensions were used to order the net, all component networks would have to be updated sychronously. If on the other hand nets were mutually referenced to the same physical points in space (sort of a neural network version of a blackboard architecture), they could learn to interact with each other. Thus one question of considerable interest was whether or not the network could learn effectively using only a subset of the possible dimensions, namely 3-D point coordinates.

For example all seven dimensions could be used to determine the distance metric or only a subset. In the present case only three of the 10 possible points were used, i.e. the xyz coordinates. Even though this meant that learning was determined by a lower dimensional projection of a higher order space, the generalization performance was very good, hence the model was capturing the problem space. It is interesting to note that this reduced dimensionality potentially permits learning for high degree of freedom robot systems, since it appears that map orderings established on metrics defined on only three dimensions were able to correctly order the values for the remaining dimensions associated with the camera and joints To understand how this would work in practical applications it is helpful to consider the sequence of steps required to generate a real-world training pattern.

A set of random joint angles is fed to the robot arm. As a result of those angles the tip of the arm moves to a location. The location is measured in terms of xyz coordinates. These coordinates are observed by the cameras which in turn produce four projection coordinates on their image planes. Ten values, the xzy positions, four camera coordinates and three joint angles then become a single member of the training set. The process is repeated for however many points are desired. The neural controllers are presented with these patterns and required to adjust their initial random values according to whatever learning rule is being used. In the present case it is the proportional winner rule.

Table 4 and Figures17a and b illustrates the performance of what happened with alternative SOM network architecture, specifically a network with a fixed 27 point neighborhood when using the proportional winner rule. Figure 18 shows a point projection of the learned network how the xy projection varied as a function of the number of data

SOM WITH 27 NEIGHBORS

Performance As A Function of
Neighborhood Size and Number of Points

(125 Points)

	1	3	4	6	20
Mean	.0932	.0974	.0909	.1082	.1254
St.Dev.	.0537	.0562	.0532	.0641	.0671
Max.	.2756	.2946	.3182	.2912	.4274
Min.	.0067	.0059	.0082	.0085	.0124
Median	.0820	.0889	.0804	.0983	.1167

(729 Points)

	1	3	4	6	20
Mean	.0418	.0459	.0471	.0488	.0586
St.Dev.	.0324	.0296	.0334	.0322	.0354
Max.	.2028	.1566	.2057	.1830	.1765
Min.	.0000	.0018	.0025	.0070	.0084
Median	.0343	.0386	.0413	.0413	.0049

(343 Points)

	1	3	4	6	20
Mean	.0646	.0618	.0626	.0692	.0915
St.Dev.	.0401	.0360	.0362	.0388	.0472
Max.	.2177	.1997	.1932	.1940	.2942
Min.	.0000	.0032	.0074	.0072	.0124
Median	.0582	.0567	.0527	.0647	.0861

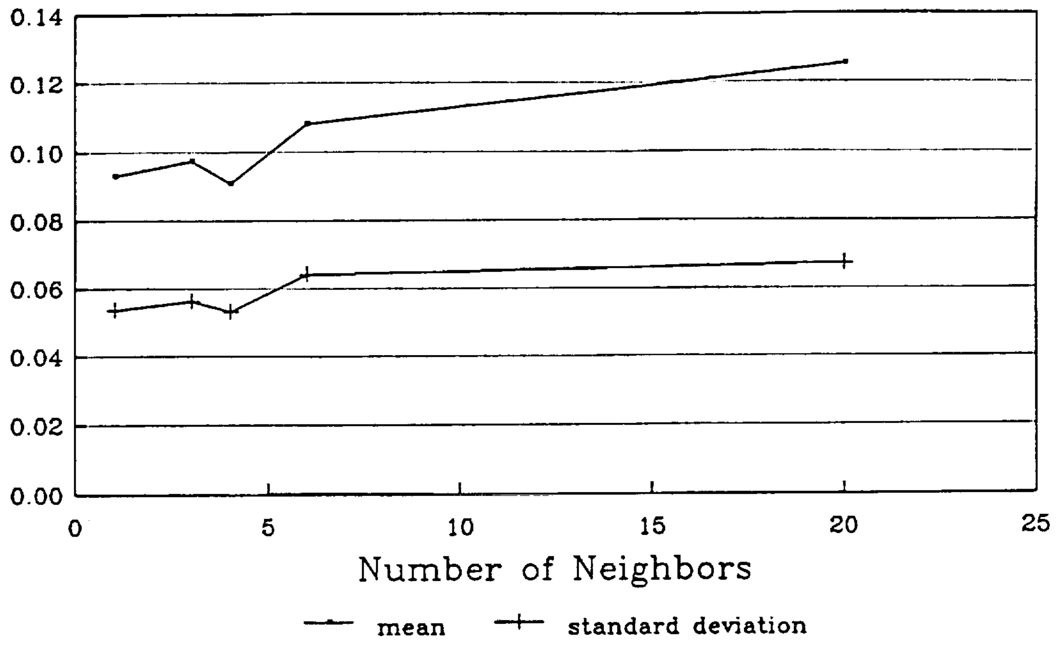
(1000 Points)

	1	3	4	6	20
Mean	.0452	.0497	.0506	.0551	.0628
St.Dev.	.0364	.0340	.0367	.0389	.0402
Max.	.1984	.1638	.2380	.2061	.2379
Min.	.0000	.0031	.0037	.0027	.0043
Median	.0358	.0417	.0404	.0468	.0560

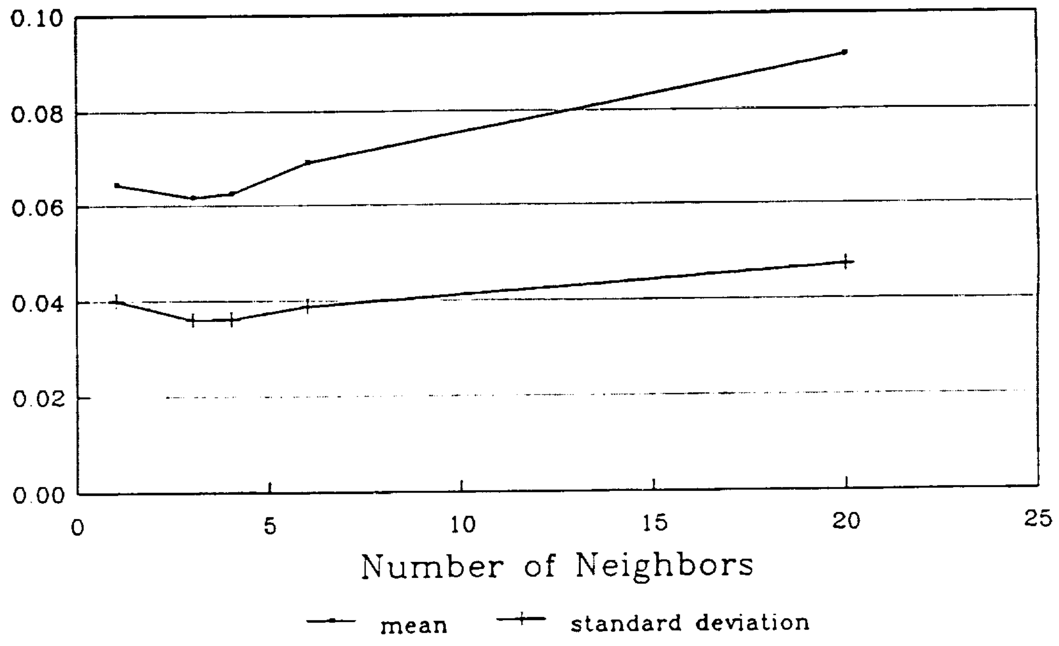
Table 4

SOM With 27 Neighbors

Performance As A Function Of
Neighborhood Size and Number of Points



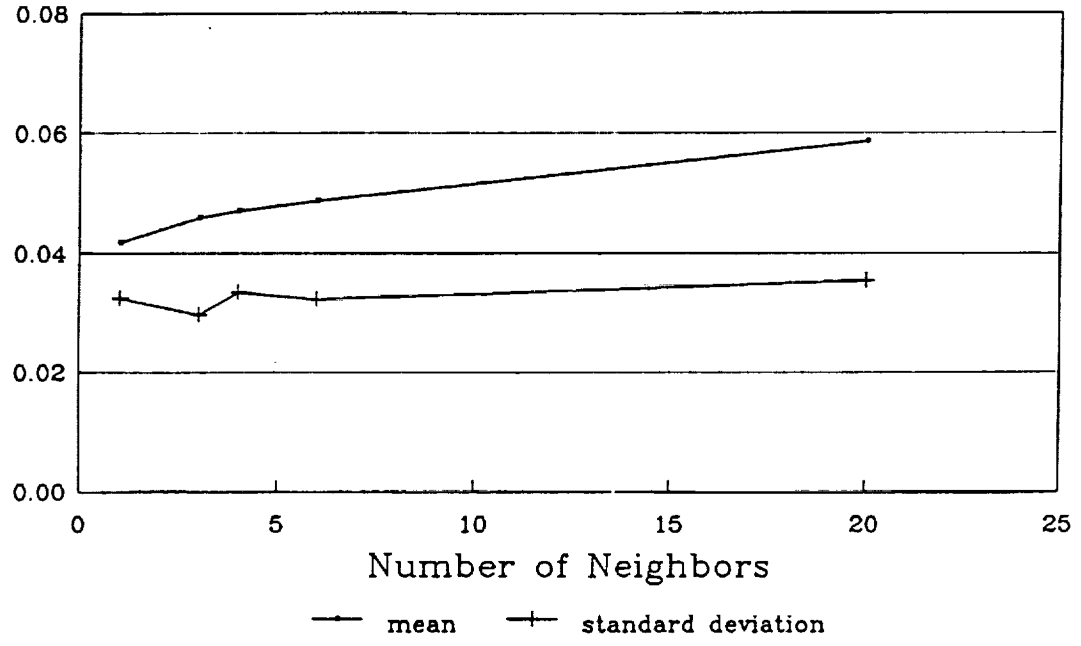
*125 data points



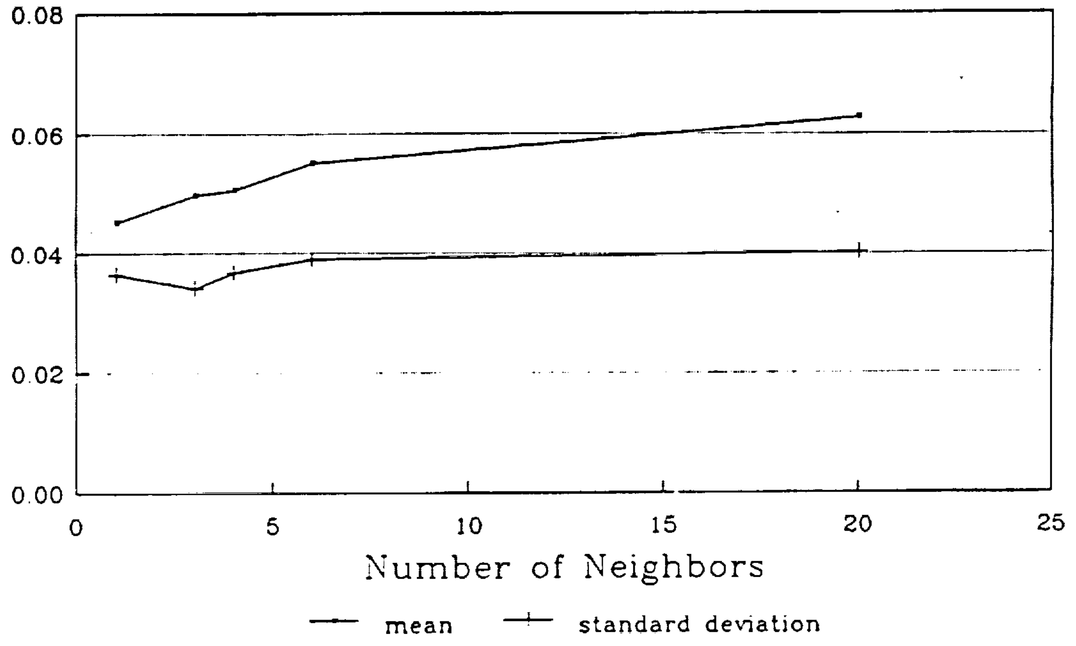
*343 data points

Figure 17a

SOM With 27 Neighbors
Performance As A Function Of Neighborhood Size and Number of Points



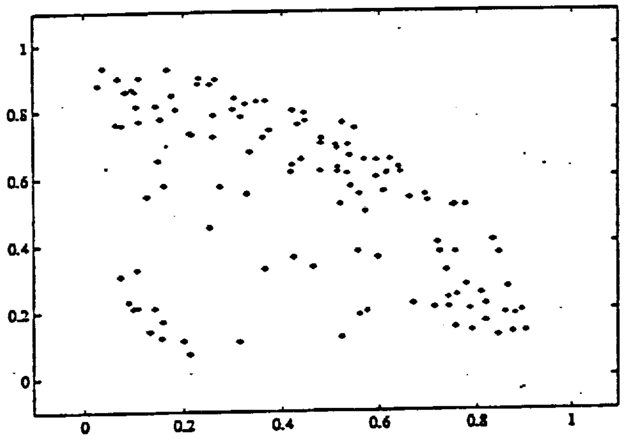
*729 data points



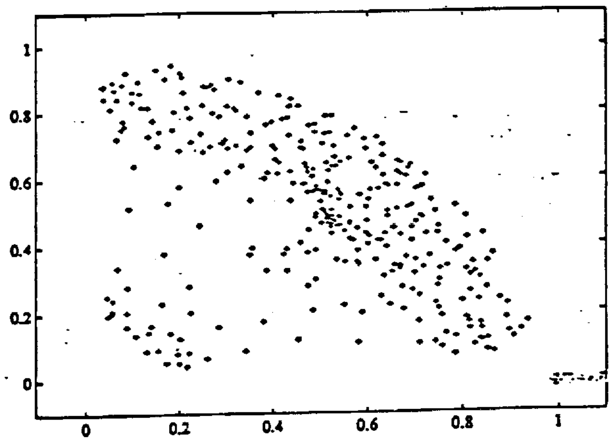
*1000 data points

Figure 17b

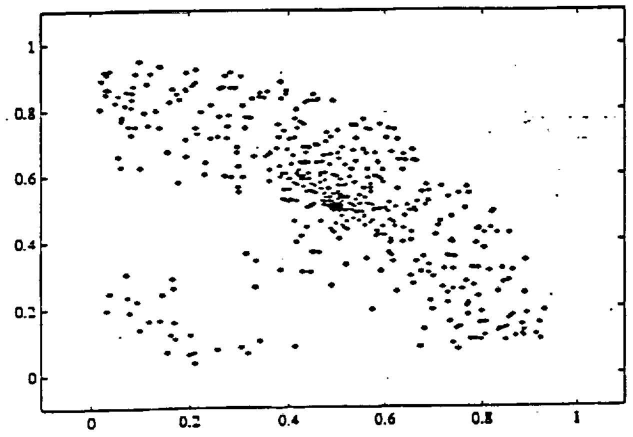
Learned Point Distribution For
27 Neighbor SOM As A Function
Of Number of Points



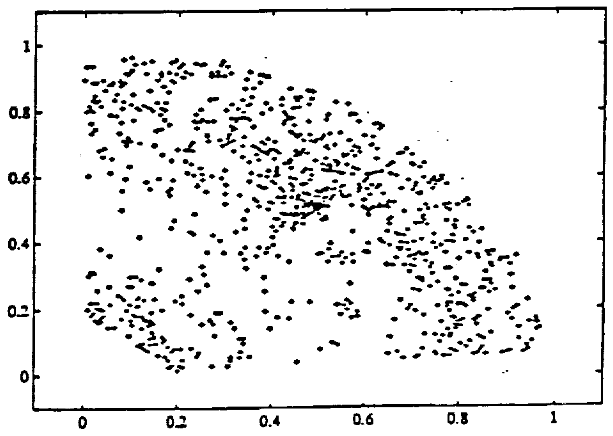
125 Points



343 Points



729 Points



1000 Points

Figure 18

points. As can be seen, the spread of points was much smoother than the underlying data distribution. After many more training trials than the previous net, generalization performance proved slightly better.

Prior to using such "hard wired" neighborhoods for the full-scale problem, the proportional winner rule was tested on the 2-D problem using two different fixed neighborhood architectures. The first neighborhood was wired to represent up, down, left, and right relationships i.e. a symmetric four member structure. Similarly, a second neighborhood added upper left, upper right, lower left, and lower right cells to produce an eight neighbor mesh. Once given a point set of arbitrary size, neighbor values were assigned at random but then were bound to those neighbors during the remainder of the learning process. Figure 19 shows the 2-D neighborhoods used. As a result of these tests, it was determined that the most densely connected neighborhood (8 neighbors) provided greater ordering stability for the 2-D mesh problem. Consequentiality a 27 neighbor mesh was constructed when the network was tested for the 3-D problem. The 27 neighbor model is found in the appendix in routinesTCJSOM3.M AND CJSOM3.M.

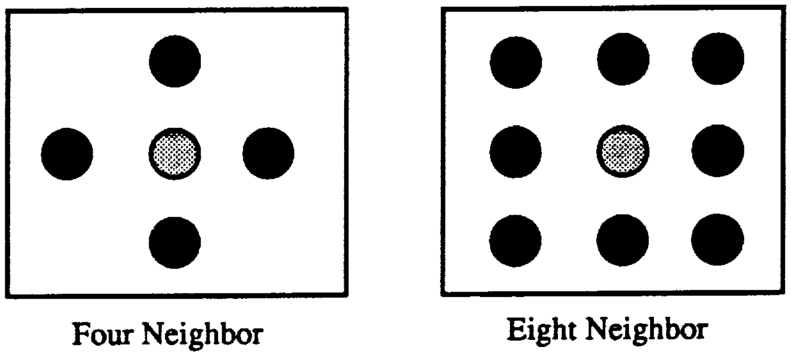


Figure 19.
Two Alternative Fixed Neighborhoods for the 2-D Problem

In the first case of radius-based neighborhoods, coding was simplified because a neighborhood was merely the n closest points. The computer overhead was also lower because a rank ordering of point distances was already being calculated during the finding of the nearest neighbor for the proportional winner rule. In most practical systems this number would generally be kept low, since most calculations for distant points would involve such small movement that distant point computations would be virtually wasted. The main effect of large learning neighborhoods was a more rapid spreading out of the points over the map. Once this spreading had occurred, significant adjustments became increasingly local in flavor until one point in the neighborhood either matched the incoming address exactly or was so close as to be unimportant.

Mean, standard deviation, minimum error, maximum error and median for the 27-neighbor net are presented in tabular form in Table 4. As a basis of comparison, performance for a nearest neighbor recall rule using the exact training set is also given. (Note: this information would never normally be available to the SOM so this is an extremely tough comparison standard. What these graphs and tables show is that a SOM network with the proportional winner rule and fixed neighborhood is capable of effectively learning the control problem and generalizing to new points in the space. The results indicate that the fixed neighborhood out-performed nearest radius learning neighborhoods for interpolation to new points for medium to large point sets. This is a valuable result since it indicates that there was a performance advantage associated with the use of the network that surpassed what a look-up table based on the training set alone would have accomplished.

A conjecture is that this result occurred because a fixed neighborhood topology ultimately permitted a more regular distribution of the global elastic energy i.e. the fixed neighborhood regularities came closer to generating a global optimum solution for the problem due to propagation of errors over may learning trials. This in turn dropped the mean statistics during the sample tests. A future research direction would be to examine machine-efficient mechanisms for the generation of higher dimensional neighborhoods. Although the 27 neighbor network was effective for this problem because of the underlying 3-D dimensional task, it may become impractical if the problem is scaled up. Consequently, the nearest radius neighborhood model tested first should not be dismissed as viable candidate even though it did not appear to perform as well in this specific case as a fixed neighborhood. The advantage appeared to remain across numbers of network points ranging from 125 to 1000.

Although effective, the large neighborhood size and bookkeeping of the fixed connected network could be a real disadvantage for real-time performance. Consequently, the next series of experiments explored a method for improving the first nearest neighborhood approach called the infolding network. This network was designed to maintain a scalable neighborhood structure and at the same time improve the performance of nearest neighborhood model for recall of previously experienced patterns, i.e. overcome the disadvantage that the SOM networks have relative to the look-up properties of the RBSDM.

Infolding Network

Although learning using a proportional winner rule and variable neighborhood was much more autonomous than learning with a standard SOM, certain characteristics of variable neighborhoods were unappealing. These were associated with ineffective use of the number of allocated data points. For example, if many points were initialized in a small cluster and allowed to expand (as is required with a hard-wired neighborhood) a number of the points would be left in the center largely unused This effect occurred because after an initial amount of spreading out, the required number of points for a nearest neighborhood were available without taking some points from the center cluster. After that stage, learning reorganized rather than distributed the remaining points. The problem did not occur with a fixed neighborhood because the points are fully interconnected across the entire grid so that eventually elastic energy minimization pulls all points apart; however, the fixed neighborhoods were much, much slower to train.

INFOLDING NETWORK

(125 Points)

	1	3	4	6	20
Mean	.0885	.0839	.0767	.0800	.1232
St.Dev.	.0547	.0536	.0447	.0492	.0666
Max.	.2695	.2734	.2164	.2384	.3924
Min.	.0061	.0000	.0071	.0077	.0204
Median	.0815	.0719	.0693	.0716	.1054

(729 Points)

	1	3	4	6	20
Mean	.0644	.0466	.0511	.0555	.0588
St.Dev.	.0479	.0327	.0382	.0402	.0401
Max.	.2673	.1895	.2237	.1998	.1940
Min.	.0061	.0000	.0000	.0042	.0032
Median	.0495	.0400	.0404	.0450	.0484

(343 Points)

	1	3	4	6	20
Mean	.0798	.0614	.0603	.0616	.1232
St.Dev.	.0540	.0403	.0378	.0379	.0462
Max.	.2604	.2515	.2536	.1930	.2391
Min.	.0061	.0027	.0000	.0038	.0071
Median	.0600	.0528	.0520	.0523	.0701

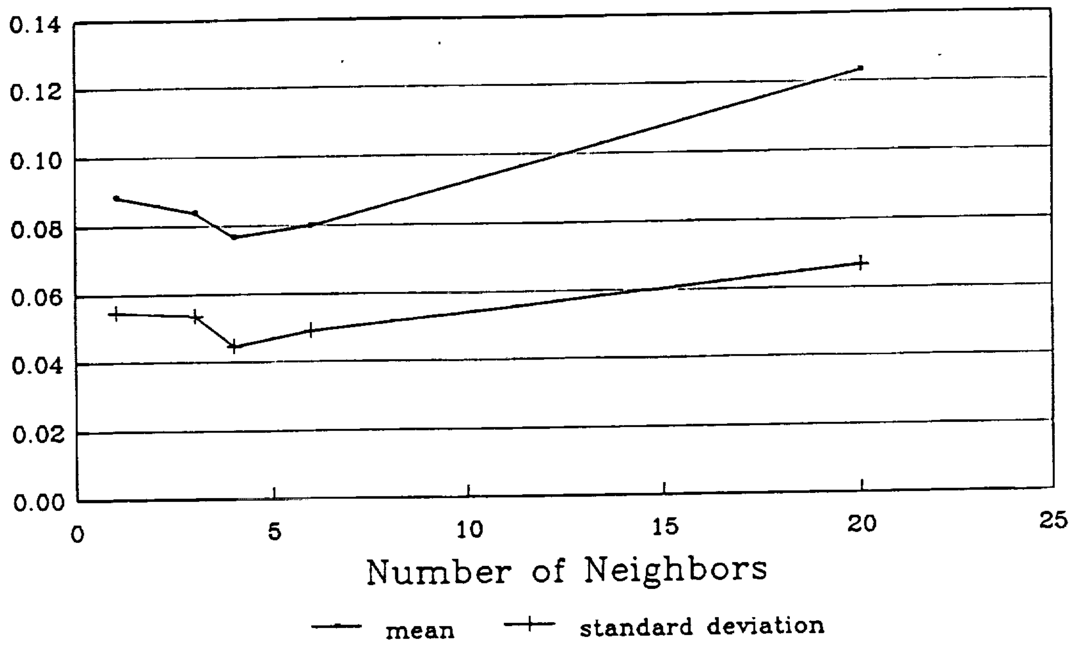
(1000 Points)

	1	3	4	6	20
Mean	.0549	.0530	.0485	.0549	.0546
St.Dev.	.0329	.0381	.0349	.0419	.0344
Max.	.2097	.1893	.1890	.2079	.1648
Min.	.0061	.0000	.0000	.0000	.0025
Median	.0502	.0426	.0384	.0451	.0438

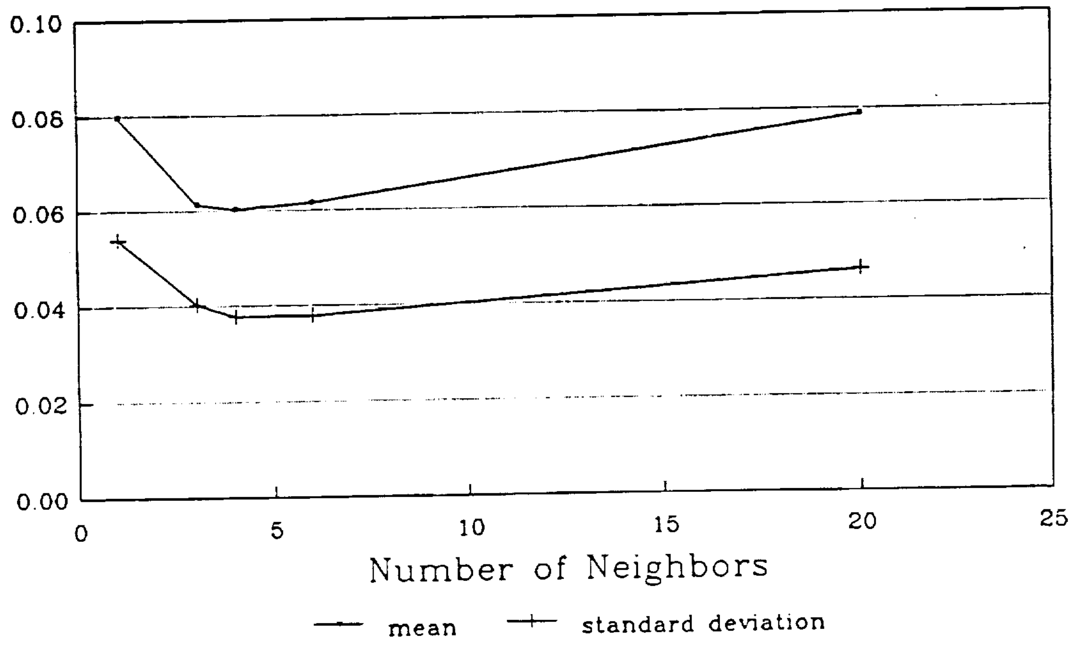
Table 5

INFOLDING NETWORK

Performance As A Function Of Neighborhood Size and Number of Points



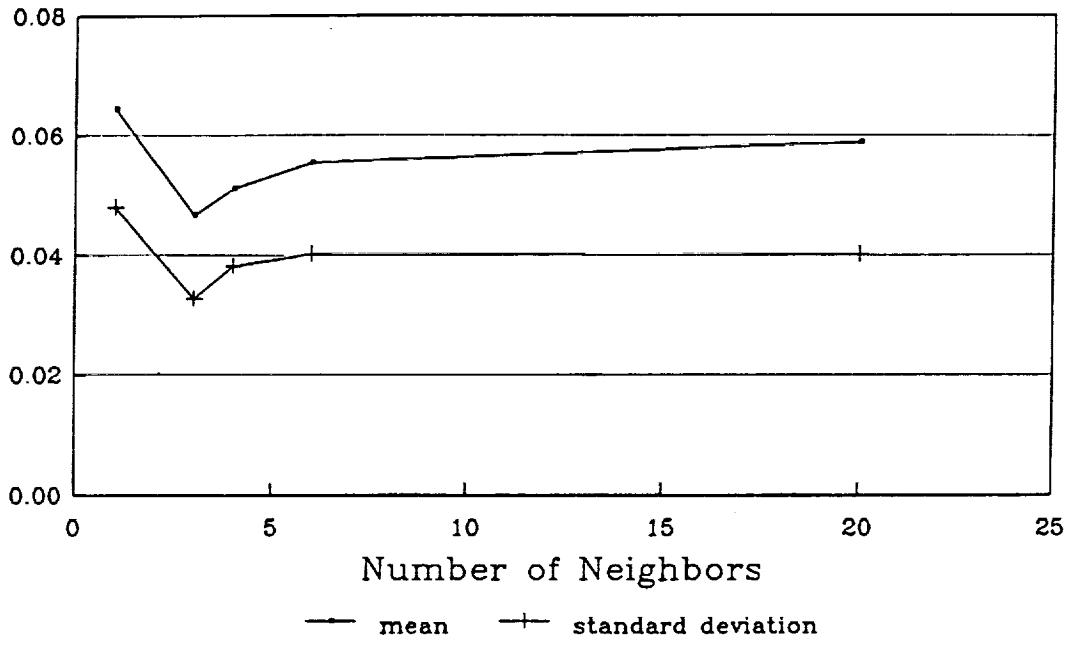
*125 data points



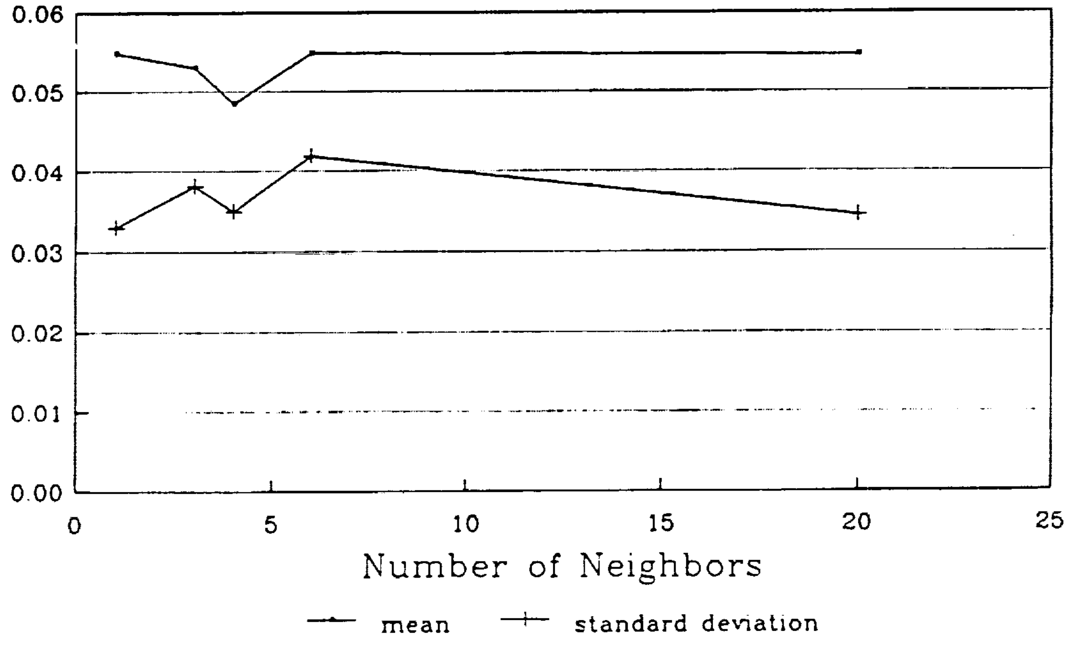
*343 data points

Figure 20a

INFOLDING NETWORK
Performance As A Function Of
Neighborhood Size and Number of Points



*729 data points



*1000 data points

Figure 20b

On the other hand, if the neighbors are not fixed and the starting points spread uniformly over all possible locations in space during initial learning (as could be done with the ranked nearest neighbor variation but not the fixed neighbor network), some points were always outside the range of legal values and were never adjusted at all. What was needed then was a network which when started with random points "swept" the multidimensional address space clean in unused areas and focussed learning on addresses in which the training process was active, the same goal that originally motived the compacting network.

One approach to this problem was what I will refer to as an Infolding Network. The logic behind the network is to use the proportional winner rule for point location adjustments and combine it with an additional process for the migration of unused points in the address space. The initial approach was to select points "farthest away" from the winning point and place them in the locations to which local neighbors would have been moved after the adjustment rule was applied. The actual nearest neighbors then remain in the same location but the density of the map is increased due to the import of more local point coordinates. When repeated, the effect of this operation was to fold the network in on itself in such a way that extreme values were constantly being removed and translated to locations reflecting local activity in the network.

Application resulted in somewhat mixed results. The majority of outlying points were indeed quickly swept out of the space and the density of points in the learning areas was increased. Unfortunately, valid but extreme data points which were not as dense tended to disappear and the map rounded and smoothed. In cases with many neighbors, the effect was to contract the net into a hypersphere.

The solution to this problem was to infold stochastically. Instead of taking only the extreme points, a point would be selected at random from all addresses in the net. The result of this process was a very rapid self organizing structure. Characteristic of this model was learning approximately two orders of magnitude faster than the connected neighborhood using the proportional winner rule. Table 5 and Figures 20 a and b, show the performance statistics for this net. Mean error of generalization for a 1000-point net was .0485 with standard deviation of .0349 using a recall neighborhood of four. This performance was superior to the best performance of RBSDM in both mean and standard deviation and was better that the four nearest neighbor performance test using the exact training data. It even exceeded the performance of the best fixed neighborhood network. Not only was recall superior, training time was almost two orders of magnitude faster! Of the approaches tested, the infolding net appeared to have the greatest potential for practical application. The use of only local neighborhoods implies that the network can be implemented in analog hardware much as Grossberg's Adaptive Resonance Models. The proportional win rule minimizes the need for external intervention and tuning in the learning process. The SOM architecture permits easy adjustment in the degree of resolution simply by changing the number of estimating points. The similarity to traditional SOM models implies that the method could be utilized for the same general problems. In summary the network exhibited the following desirable characteristics:

1. It rapidly learned a training set and readjusted automatically if the set changed in

time.

2. It was able to deal with arbitrary resolutions defined by the number of estimation points and to interpolate smoothley over the training surface.

3. It did not require fine tuning of alpha, radius, and apriori knowledge of the process.

4. It avoided the need to compute fixed neighborhoods which was of particular advantage if the number of dimensions increased.

5. It did not require a euclidean distance metric such as that used in these experiments.

6. Learning occurred an order of one to two magnitudes faster than without the infolding behavior. This was largely due to a rapid removal of inactive addresses.

7. It addresses the stability plasticity-dilemma by the random replacement of previously learned addresses with new information. The degree of plasticity of the memory automatically increases when more learning is required and slows or freezes if previously learned points are presented.

8. Further, the stochastic replacement behavior implements a form of birth and death process. Unused addresses are gradually swept away by the relocation of points. The moved points function much like new regional connections because the next time the same area is accessed, a more compact neighborhood becomes available. The dark star behavior is avoided because training points from other areas of the map constantly reallocate to conform to the current distribution of the learning set. If the training process degenerates into the same point for thousands of trials, the net would not collapse into an attractor because the coefficient of the proportional win rule would go to zero and stop further evolution of the net.

The point about birth and death processes requires additional comment. By way of review, the plasticity-stability dilemma refers to the need of a real-time process to be able to learn and recall from a memory simultaneously. That is, the process can not stop and go through an extended learning process after each new data point enters the system. Because the network above stores the latest point due to the proportional winner rule, recent information is available for look-up. Unless it is refreshed, older information is removed and the addresses are used to fill in areas of greater activity. The difference in the learning rate (very fast) versus the forgetting rate (slow if the number of points is large) permits handling this problem in an elegant fashion.

Discussion

Comparisons of the generalizability of the training data to the test set show that the best level of learning performance occurred with the infolding network, exceeding that of the RBSDM for new patterns. A summary comparing the performance between the infolding net, the fixed neighborhood SOM and the RBSDM is given in Figure 21.The infolding network will not equal the performance of RBSDM for previously encountered patterns unless the neighborhood is set to one. This is because the RBSDM functions as a look- up table and has not physically migrated the locations of the training point coordinates during learning. If the task had only a small number of potential states and a non time-varying process model, the RBSDM could be a better choice. If on the other hand, it is not known a priori that the function being modeled is smoothly continuous, or the process changes with time and requires readjustment, the Infolding Network appears the most desirable choice

Regarding the most effective method for capturing the underlying characteristics of the generating function, the infolding network appeared the most efficient of the methods tested in that if the neighborhood of influence was set to one and the number of points in the memory equalled the number of points in the training set, the network would capture the exact structure of the training set. If however, the neighborhood of influence was increased and/or the number of points decreased, the network moved point locations in such a way as to estimate an average point location of the set. If the number of approximating points was small and the radius of influence small, the network would track the trial-to-trial variations in the training set samples i.e. it became a time-varying network structure.

A second concern was the best way to use the captured information when estimating a new point. In this study, a four-neighbor variation on a k-nearest neighbor rule showed the best recall performance evidenced by the dip in mean error between 3 and 5 on the majority of the network plots. The dip was similar for data point ranges from 125 to 1000 and was mirrored by corresponding improvements in the standard deviations for all but very large numbers of points. As was mentioned in the discussion of the RBSDM, alternative recall strategies were tried without significantly improving performance. One was to form the estimated camera and joint coordinates based upon an average of the weighted gaussian distances from the input value. This weighted average did not perform as well as a simple four nearest neighbor average. From the standpoint of machine implementation, the fact that the simplest algorithm appeared to have the best performance is desirable. The simplicity of calculation combined with a small local neighborhood bodes well for the scalability of the algorithm.

During this research, observations were made which have implications for further study. One of these concerned the effects of varying the size of the neighborhood when using the proportional winner rule. For example, if the neighborhood was over ten percent of the total number of points, a behavior analogous to a skeletonized image processing was observed. The large neighborhood resulted in disconnected clusters which moved to points that had the greatest frequency in the training sample. Within the cluster, points

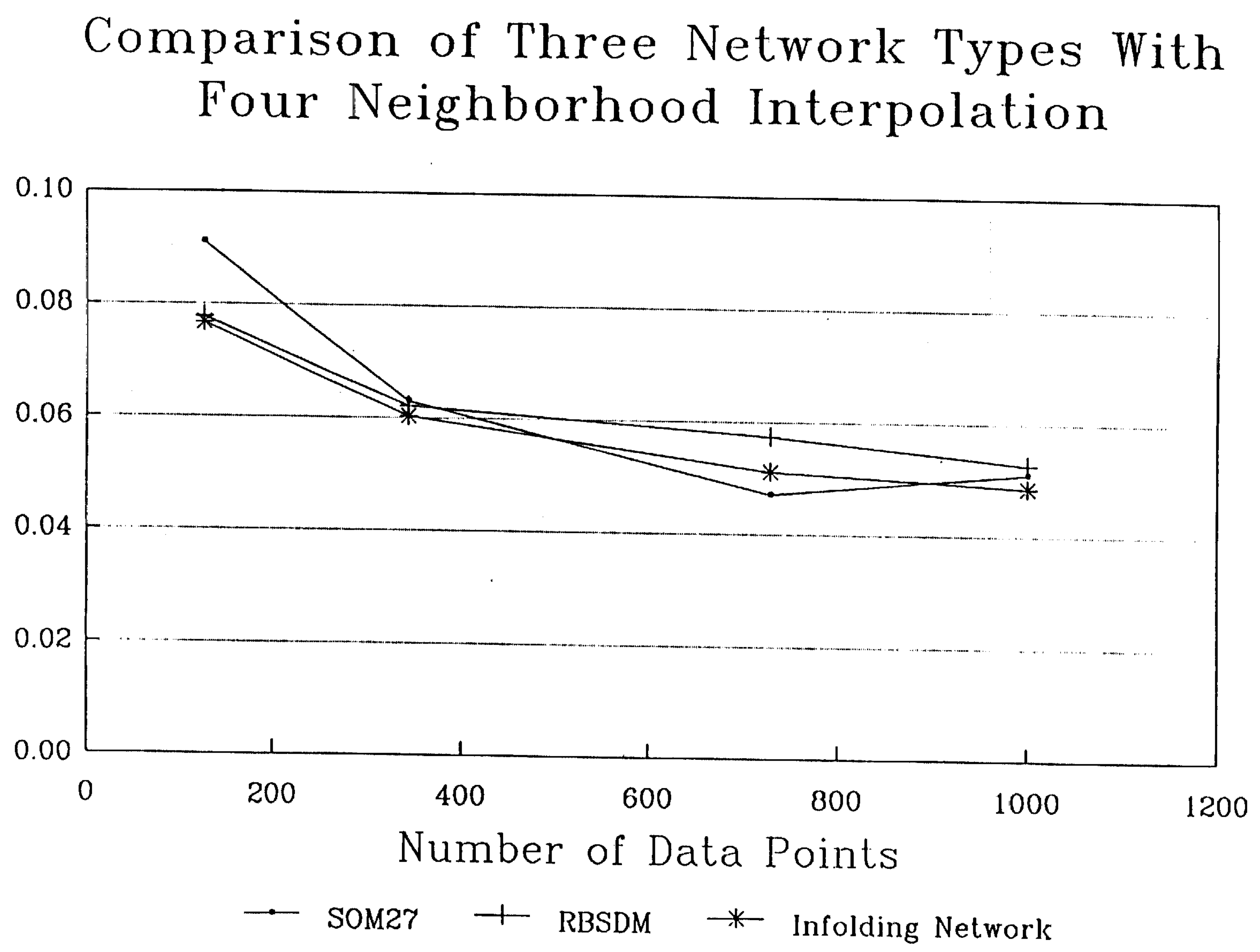

Comparison of Three Network Types With
Four Neighborhood Interpolation
0.10
0.08
0.06
0.04
0.02
0.00
0
200
400
600
800
1000
1200
Number of Data Points
SOM27
RBSDM
Infolding Network

Figure 2 1

remained tightly grouped providing a coarse outline of the underlying distribution. There appeared to be a similarity between such clusters and moments of the generating training distribution. Given the ability of the infolding net to track time-changing distributions, the simultaneous use of multiple networks using different neighborhood sizes might permit improved approximations to a generating distribution. The maps might also capture different time scales because a given random replacement rule would have more of an impact for small point sets (coarse grain) than large sets (fine grain). The implication for further study is that temporal patterns might be coded through the use of multiple or time windowed copies of the networks. Most current research on SOM maps and distributed memories has focused on finding a single optimal neighborhood and recall rule without considering the possible utility of parallel multiresolution self-organizing map structures for temporal pattern matching.

However, the clearest future direction lies in increasing the complexity of the problem to a higher degree of freedom manipulator. The benchmark was constructed in such a way that a natural path has been provided for more sophisticated tests, namely through the addition of arm and joint segments, force and computed torques, and enhanced camera or sensor models. Given that the infolding network appeared most successful on the current problem, the next logical test would be the substitution of actual arm hardware. Systems capable of this type of testing are available at both Ames and Langley research centers. An interesting enhancement would be to couple the distributed memory logic of the enfolding network to enhanced computing architectures specifically, photonic or massively parallel processes.

A final comment concerns the incorporation of human control functions using the models developed during this study. Because the SOM variations in the present paper have maintained many of the distributional characteristics of the RBSDM, it is possible to analyze individual point addresses and reconstruct specific behaviors for given state conditions (i.e. input addresses). However, because the network will also average information and adjust the point locations depending upon the influences of the entire training set, a clear separation of formally generated control information and that produced by a supplementing human controller would be lost. The gain would be a smoothly integrated network combining both sources of information but with a higher variance than a formal model alone. If what is desired is a look-up table behavior for precisely defined control responses and an interpolating table for human or supplementary learned modifications, the RBSDM framework may be the most effective for this specific goal because sudden changes in state behavior are not subject to interpolation. It's disadvantage is a much less flexible capability for handling reduced numbers of data points, inability to track time-changing phenomena, and poorer generalization for smaller points sets. Such sets may well occur do to limited availability of samples.

Appendix

MATLAB Source Code For Neural Controllers Discussed In Paper

```
function [outval,numcat] = pnn(ncats,maxy,inpat,numpat,Examples,nsamples,sigma)

%
% Matlab routine  to calculate a probabalistic neural network
% category number and value in response to an input vector called inpat

% ncats is the number of categories into which output has been divided
% maxy is the largest value in the output value set i.e.  (y(x))
% Previous observations are stored in a matrix called Examples
% numpat is the number of input variables or dimensions
% and is the same as the dimension of the Examples
% Note: Examples is a matrix, the last column are the correct categorie
% the first numpat columns are the sample input variable values
% thus for a one input, one output equation the first column is x
% the second column is the category for y
% nsamples is the number of examples stored in Examples
% Interval size must be input, it is max y divided by ncat

% sigma is the deviation value for the gaussians around a data point
% it is squared and depends upon the coarseness required for category
% resolution (try .8 as a start but remember for the PNN to work
% correctly this must be close to correct. Trial and Error usually finds
% it faster than theory in my experience!

% Begin Routine ******************

% Calculate sums of inpat gaussians in terms of deviation from Examples

gaussums=(exp(-((Examples(:,1:numpat-1)-inpat).^2)./(sigma^2)));
clg;
axis('normal');
subplot(211), plot(gaussums),grid;
title(' Gaussian Distances From Examplars ');
xlabel('sigma=.1, 400 examples, 50 categories')
% Total sums of Gaussians for each category, select maximum and its index

outsum=zeros(1:ncats);
for i=1:nsamples
outsum(Examples(i,numpat))=outsum(Examples(i,numpat))+gaussums(i); end

axis([1,ncats,min(outsum),2*max(outsum)]);
subplot(212),plot(outsum),grid;
title(' Total Sums of Gaussian Values by Category Number '),pause;
axis;

print;
% Send hard copy to local postscript printer
[outval,numcat]=max(outsum)

% Completed PNN recall now compare to correct value ************

% Place to put an equation to compare actual to estimated values
% This would be for example an F-15 equation for one part of the
% controller, I put a simple linear one here y  = .6x
% I used y increments of .03 for PNN category widths

estval=(numcat/ncats)*maxy
actval=.6*inpat
```

```
function Examples=catdat(yvector,numex,ncats,xmat)
intsize=max(yvector)/ncats;
for i=1:numex
    for j=1:ncats
if ((yvector(i)>intsize*(j-1))&((yvector(i)<=intsize*j)))
  Examples(1,i)=xmat(i);
  Examples(2,i)=j;
end
end
end

% Generic Function to put input data into categories for Examples
% y is the input matrix, the second row of which contains the category
% index corresponding to the column index of Examples. This lets
% data be input in any category order. There is one row for each
% variable used in inpat. xmat is the input variable matrix from
% which some function has calculated y values provided to catdat
```

```
function ov=sdmhybrid(na,wa,wd,ia,A,D,c,hrange,sigma,lrncnt)
% Implimentation of a gaussian hybrid sdm for control
% na = number of addresses used in the sdm (# previous patterns)
% wa = width of input address vector i.e. # of input variables
% wd = width of data for output i.e. # of output variables
% A is address matrix, na rows by wa columns - stored input examples
% D is a matrix of data na rows by wd columns - stored system output values
% ia = input address of new pattern - wa input values
% lrncnt is count of frequency of previous examples in active cells
%      it may not be used unless distributional information is important
% ov = network's output responses to the new pattern
% gausdist = vector of distances of ia from A (distance defined as gausian)
% hrange = percent of maximum gaussian value (larger means closer match)
%    the only values that will be used are greater or equal to  hrange*max
% sigma controls the width of the gaussians
% A can be either random addresses or fixed exemplar points in parameter space
%    to emulate a pnn in this case I chose the latter
% c is a counter vector storing the number of matches for an address in D
%    this occurs with noise if there is more than one example per address
% You must build A, D and c outside this function in matlab at present
%  this assumes previous examples stored in A with values D

sigma2=sigma^2;
% calculate distance of input pattern to gaussian window around previous
% stored patterns total the distances and store in gausdist
for i=1:na, gausdist(i)=sum(exp(-(((ia - A(i,1:wa)).^2)./(sigma2)))); end
% plot(gausdist),title('Gaussian similarity to A'),pause;
% print;
% Make hrange proportional to max range found by gausdist
hrange=hrange*max(gausdist);
% hrange
%convert gausdist to 0,1 selection matrix stored in D
gausdist=(gausdist>=hrange);
% gausdist
% pause;
% plot(gausdist),title('Selection distribution from 0,1 transform'), pause;
% update counter weights for selected D values if lrn-rcl is learn (+1)
if (lrncnt==1) c= c + gausdist; else
numc=gausdist .* c;
% Note c can be a vector stored from previous learning counters
% Here is is being used as a scalar only
% plot(numc),title('Third plot showing vector numc'),pause;
ov=(numc * D)./sum(numc);
% ov
% pause;
end
```

```
function status3dsom=tcjsom3(trainset,startpts)
% Routine to test a SOM performance on a random 3D grid. It is also a demo
% of the adequacy of the network's alpha and r values for a training set
axis([-.1 1.1 -.1 1.1]);
% Use this block only if you want a random set of points and plot them
% Unless you want to change the internal grid dimensions use even square
% root number of points
clg;
% this ignores the startpts argument passed to the routine in order to
% build a new set
%startpts=[];
% This part of the code lets you build a distributed random set of points
% for i=1:10 startpts(i,:)=[rand rand rand rand rand rand rand rand rand rand];
%end
% Use this block if you want the points to start at random in a center region
% Numpts should have an even cube root otherwise modify code
%numpts=1000;
%for i=1:numpts
% startpts(i,1)=(.49+rand*.02);
% startpts(i,2)=(.49+rand*.02);
% startpts(i,3)=(.49+rand*.02);
% startpts(i,4)=(.49+rand*.02);
% startpts(i,5)=(.49+rand*.02);
% startpts(i,6)=(.49+rand*.02);
% startpts(i,7)=(.49+rand*.02);
% startpts(i,8)=(.49+rand*.02);
% startpts(i,9)=(.49+rand*.02);
% startpts(i,10)=(.49+rand*.02);
% end
temp=length(startpts);
temp2=length(trainset);
% Show the starting x-y plot to get some idea if ordering is occuring
plot(startpts(:,1),startpts(:,2));
% Set parameter values for alpha and the radius
count=1;
a=.08;
r=.2;
patnum=1;
% set the number of interations at each alpha and radius
for j=1:40000
if (patnum==temp2) patnum=1; else patnum=patnum+1; end
count=count+1;
[nvec,est,wv]=cjsom3(trainset,startpts,r,a,0,patnum);
% Set ratio of decreases for learning rate and radius
a=a*.99999;
% r=r*.99999;
startpts=nvec;
% startpts
% pause;
% Set how often you would like to see the plot in plotsee
plotsee=100;
if(((count/plotsee)-round(count/plotsee))==0)
plot(nvec(:,1),nvec(:,2),'+');
% plot(trainset(:,1),trainset(:,2),'*');
% Use these if you want to see how the radius or alpha rate is changing
count
a
end
end
%hold;
% Plot a comparison of the final results to the ideal results for 100 pts
% plot(trainset(:,1),trainset(:,2),'x')
status3dsom=nvec;
%hold off;
```

```
function [ov,m,wv]=cjsom3(pvec,changevec,bubble,lrnrate,testcode,pn)
% Implementation of a 3-D Self Organizing Map for learning Camera-Theta
%  relations. Uses mesh defined above, below, left, right, front,back for
% each point in changevec
% if testcode is 1 it returns the net's estimate of the winner other wise
% if 0 it treat's  pvec as a training pattern
% lrnrate learning rate (proportion of change of previous point values)
% ov = network's structure reconfiguration in response to the new pattern
% dist = vector of distances of patterns from input
% bubble controls the width of influece field of a neuon
ov=changevec;
% Put in a couple checks on the size and value of lrnrates
if lrnrate<0 lrnrate=-lrnrate; end
if lrnrate>1 lrnrate=1.0; end
npl=length(pvec);
% initialize variables
na=length(changevec);
% select a pattern newp from pvec at random
newp=pvec((ceil(rand*npl)),:);
% Use this line if you want to train using all patterns in order
% and omit the above line
%newp=pvec(pn,:);
% calculate distance of stored patterns from new
% pattern and find closest pattern based only on xyz distances
newpvec=(ones(na,1)*newp);
diffs=newpvec-changevec;
% Note: diffs should have and index equal to the space used with more
% complex data e.g. robot arm data this will change to diffs(:,1:3)
% dist=sqrt(sum((diffs(:,1:3).^2)'));
dist=sum((diffs(:,1:3).^2)');
[wv,winnum]=min(dist);
% if network is only to estimate a value return it in m and stop
if (testcode==1) m=changevec(winnum,:); return; else m=0;  end
% [lv,lnum]=max(dist);
% Use these lines if you want to change all points less than bubble each trial
%for j=1:na
%     if dist(j)<=bubble
%winnum=j;
% create links from points to their neighbors to generate a mesh
vecsize=floor((na)^(1/3));
% Note vector should have an even root or explicitly be put in somvectomat call
% change the following x,y and z dimensions accordingly
% refpos is the x,y,z reference coordinate around which neighbors are defined
refpos=somvectomat(winnum,[vecsize vecsize vecsize]);
% Define the 26 coordinates for the neighbors relative to refpos
% Define center neighborhood of a point
uc=[(refpos(1)) (refpos(2)) (refpos(3)+1)];
dc=[(refpos(1)) (refpos(2)) (refpos(3)-1)];
lc=[(refpos(1)-1) (refpos(2)) (refpos(3))];
rc=[(refpos(1)+1) (refpos(2)) refpos(3)];
ulc=[(refpos(1)-1) (refpos(2)) (refpos(3)+1)];
urc=[(refpos(1)+1) (refpos(2)) (refpos(3)+1)];
dlc=[(refpos(1)-1) (refpos(2)) (refpos(3)-1)];
drc=[(refpos(1)+1) (refpos(2)) (refpos(3)-1)];
% Define front neighborhood (relative to the center point)
uf=[(refpos(1)) (refpos(2)-1) (refpos(3)+1)];
df=[(refpos(1)) (refpos(2)-1) (refpos(3)-1)];
lf=[(refpos(1)-1) (refpos(2)-1) (refpos(3))];
rf=[(refpos(1)+1) (refpos(2)-1) refpos(3)];
ulf=[(refpos(1)-1) (refpos(2)-1) (refpos(3)+1)];
urf=[(refpos(1)+1) (refpos(2)-1) (refpos(3)+1)];
dlf=[(refpos(1)-1) (refpos(2)-1) (refpos(3)-1)];
drf=[(refpos(1)+1) (refpos(2)-1) (refpos(3)-1)];
cf=[(refpos(1))  (refpos(2)-1) (refpos(3))];
% Define back neighborhood (relative to center point
ub=[(refpos(1)) (refpos(2)+1) (refpos(3)+1)];
```

```
db=[(refpos(1)) (refpos(2)+1) (refpos(3)-1)];
lb=[(refpos(1)-1) (refpos(2)+1) (refpos(3))];
rb=[(refpos(1)+1) (refpos(2)+1) refpos(3)];
ulb=[(refpos(1)-1) (refpos(2)+1) (refpos(3)+1)];
urb=[(refpos(1)+1) (refpos(2)+1) (refpos(3)+1)];
dlb=[(refpos(1)-1) (refpos(2)+1) (refpos(3)-1)];
drb=[(refpos(1)+1) (refpos(2)+1) (refpos(3)-1)];
cb=[(refpos(1))  (refpos(2)+1) (refpos(3))];
% Find out which serial coordinate the neighbors have in startpts
ucvec=sommattovec(uc,[vecsize vecsize vecsize]);
lcvec=sommattovec(lc,[vecsize vecsize vecsize]);
dcvec=sommattovec(dc,[vecsize vecsize vecsize]);
rcvec=sommattovec(rc,[vecsize vecsize vecsize]);
ulcvec=sommattovec(ulc,[vecsize vecsize vecsize]);
urcvec=sommattovec(urc,[vecsize vecsize vecsize]);
dlcvec=sommattovec(dlc,[vecsize vecsize vecsize]);
drcvec=sommattovec(drc,[vecsize vecsize vecsize]);
% front block
ufvec=sommattovec(uf,[vecsize vecsize vecsize]);
dfvec=sommattovec(df,[vecsize vecsize vecsize]);
lfvec=sommattovec(lf,[vecsize vecsize vecsize]);
rfvec=sommattovec(rf,[vecsize vecsize vecsize]);
ulfvec=sommattovec(ulf,[vecsize vecsize vecsize]);
urfvec=sommattovec(urf,[vecsize vecsize vecsize]);
dlfvec=sommattovec(dlf,[vecsize vecsize vecsize]);
drfvec=sommattovec(drf,[vecsize vecsize vecsize]);
cfvec=sommattovec(cf,[vecsize vecsize vecsize]);
% back block
ubvec=sommattovec(ub,[vecsize vecsize vecsize]);
dbvec=sommattovec(db,[vecsize vecsize vecsize]);
lbvec=sommattovec(lb,[vecsize vecsize vecsize]);
rbvec=sommattovec(rb,[vecsize vecsize vecsize]);
ulbvec=sommattovec(ulb,[vecsize vecsize vecsize]);
urbvec=sommattovec(urb,[vecsize vecsize vecsize]);
dlbvec=sommattovec(dlb,[vecsize vecsize vecsize]);
drbvec=sommattovec(drb,[vecsize vecsize vecsize]);
cbvec=sommattovec(cb,[vecsize vecsize vecsize]);
% if they are legitimate indexes of the vector use them otherwise do not
% Correct changevec by moving neighborhood points toward the new pattern
% proportional to the nearest pattern's distance
if (ucvec>0)&(ucvec<=na)
changevec(ucvec,:)=changevec(ucvec,:)+...
(wv/dist(ucvec))*(lrnrate)*(diffs(ucvec,:)); end
if (lcvec>0)&(lcvec<=na)
changevec(lcvec,:)=changevec(lcvec,:)+...
(wv/dist(lcvec))*(lrnrate)*(diffs(lcvec,:)); end
% Unique treatement for winning center point
changevec(winnum,:)=changevec(winnum,:)+...
(lrnrate)*(diffs(winnum,:));
if (dcvec<0)&(dcvec>=na)
changevec(dcvec,:)=changevec(dcvec,:)+...
(wv/dist(dcvec))*(lrnrate)*(diffs(dcvec,:)); end
if (rcvec>0)&(rcvec<=na)
changevec(rcvec,:)=changevec(rcvec,:)+...
(wv/dist(rcvec))*(lrnrate)*(diffs(rcvec,:)); end
if (ulcvec>0)&(ulcvec<=na)
changevec(ulcvec,:)=changevec(ulcvec,:)+...
(wv/dist(ulcvec))*(lrnrate)*(diffs(ulcvec,:)); end
if (urcvec>0)&(urcvec<=na)
changevec(urcvec,:)=changevec(urcvec,:)+...
(wv/dist(urcvec))*(lrnrate)*(diffs(urcvec,:)); end
if (dlcvec>0)&(dlcvec<=na)
changevec(dlcvec,:)=changevec(dlcvec,:)+...
(wv/dist(dlcvec))*(lrnrate)*(diffs(dlcvec,:)); end
if (drcvec>0)&(drcvec<=na)
changevec(drcvec,:)=changevec(drcvec,:)+...
```

```
(wv/dist(drcvec))*(lrnrate)*(diffs(drcvec,:)); end
% begin front block
if (ufvec>0)&(ufvec<=na)
changevec(ufvec,:)=changevec(ufvec,:)+...
(wv/dist(ufvec))*(lrnrate)*(diffs(ufvec,:)); end
if (dfvec>0)&(dfvec<=na)
changevec(dfvec,:)=changevec(dfvec,:)+...
(wv/dist(dfvec))*(lrnrate)*(diffs(dfvec,:)); end
if (lfvec>0)&(lfvec<=na)
changevec(lfvec,:)=changevec(lfvec,:)+...
(wv/dist(lfvec))*(lrnrate)*(diffs(lfvec,:)); end
if (rfvec>0)&(rfvec<=na)
changevec(rfvec,:)=changevec(rfvec,:)+...
(wv/dist(rfvec))*(lrnrate)*(diffs(rfvec,:)); end
if (ulfvec>0)&(ulfvec<=na)
changevec(ulfvec,:)=changevec(ulfvec,:)+...
(wv/dist(ulfvec))*(lrnrate)*(diffs(ulfvec,:)); end
if (urfvec>0)&(urfvec<=na)
changevec(urfvec,:)=changevec(urfvec,:)+...
(wv/dist(urfvec))*(lrnrate)*(diffs(urfvec,:)); end
if (dlfvec>0)&(dlfvec<=na)
changevec(dlfvec,:)=changevec(dlfvec,:)+...
(wv/dist(dlfvec))*(lrnrate)*(diffs(dlfvec,:)); end
if (drfvec>0)&(drfvec<=na)
changevec(drfvec,:)=changevec(drfvec,:)+...
(wv/dist(drfvec))*(lrnrate)*(diffs(drfvec,:)); end
if (cfvec>0)&(cfvec<=na)
changevec(cfvec,:)=changevec(cfvec,:)+...
(wv/dist(cfvec))*(lrnrate)*(diffs(cfvec,:)); end
% Back block
if (ubvec>0)&(ubvec<=na)
changevec(ubvec,:)=changevec(ubvec,:)+...
(wv/dist(ubvec))*(lrnrate)*(diffs(ubvec,:)); end
if (dbvec>0)&(dbvec<=na)
changevec(dbvec,:)=changevec(dbvec,:)+...
(wv/dist(dbvec))*(lrnrate)*(diffs(dbvec,:)); end
if (lbvec>0)&(lbvec<=na)
changevec(lbvec,:)=changevec(lbvec,:)+...
(wv/dist(lbvec))*(lrnrate)*(diffs(lbvec,:)); end
if (rbvec>0)&(rbvec<=na)
changevec(rbvec,:)=changevec(rbvec,:)+...
(wv/dist(rbvec))*(lrnrate)*(diffs(rbvec,:)); end
if (ulbvec>0)&(ulbvec<=na)
changevec(ulbvec,:)=changevec(ulbvec,:)+...
(wv/dist(ulbvec))*(lrnrate)*(diffs(ulbvec,:)); end
if (urbvec>0)&(urbvec<=na)
changevec(urbvec,:)=changevec(urbvec,:)+...
(wv/dist(urbvec))*(lrnrate)*(diffs(urbvec,:)); end
if (dlbvec>0)&(dlbvec<=na)
changevec(dlbvec,:)=changevec(dlbvec,:)+...
(wv/dist(dlbvec))*(lrnrate)*(diffs(dlbvec,:)); end
if (drbvec>0)&(drbvec<=na)
changevec(drbvec,:)=changevec(drbvec,:)+...
(wv/dist(drbvec))*(lrnrate)*(diffs(drbvec,:)); end
if (cbvec>0)&(cbvec<=na)
changevec(cbvec,:)=changevec(cbvec,:)+...
(wv/dist(cbvec))*(lrnrate)*(diffs(cbvec,:)); end
% end
% end
ov=changevec;
```

```
function [alpnext,rnext]=sofmrates(d,alp,r,startd,startr)
% Routine to calculate governing differential equations for time
% varying self-organizing map with dynamic learning of bubble size
% and oscillating learning rates based on method of Hodges and Wu (1990)
% r is neighborhood radius at time t
% alpha is learning rate at time t
% d is minimum distance between input pattern and exemplars at time t
g=d/startd;
s=r/startr;
alpnext=s^g*(exp(-s)*exp(-1/g)*(tanh(1+alp*g)/startr^2)-alp*exp(-alp));
rnext=r+(-(s^g)*(exp(-(1-alp)))*exp(-1/s)*exp((-(1-g))))-(s^2)*exp(-(s^2));
```

```
function status3dsom=tregsom(trainset,startpts, numit, netsize,nnum)
% Routine to test a SOM performance on a random 3D grid. It is also a demo
% of the Infolding Network using the proportional win learning rule
% and a stochastic point sampling logic
% numit is the number of learning trials desired
% trainset is the matrix of potential training patterns
% startpts is the set of points
% netsize is the number of points desired in startpts
% nnum is the size of the neighborhood for this network test
axis([-.1 1.1 -.1 1.1]);
clg;
% this code ignores the startpts argument passed to the routine in order to
% build a new set of a given number of points
startpts=[];
% This part of the code lets you build a distributed random set of points
%for i=1:netsize startpts(i,:)=[rand rand rand rand rand rand rand rand rand rand];
%end
% Use this block if you want the points to start at random in a center region
numpts=netsize;
for i=1:numpts
 startpts(i,1)=(.49+rand*.02);
 startpts(i,2)=(.49+rand*.02);
 startpts(i,3)=(.49+rand*.02);
 startpts(i,4)=(.49+rand*.02);
 startpts(i,5)=(.49+rand*.02);
 startpts(i,6)=(.49+rand*.02);
 startpts(i,7)=(.49+rand*.02);
 startpts(i,8)=(.49+rand*.02);
 startpts(i,9)=(.49+rand*.02);
 startpts(i,10)=(.49+rand*.02);
end
temp=length(startpts);
temp2=length(trainset);
% Show the starting x-y plot to get some idea if crdering is occuring
plot(startpts(:,1),startpts(:,2));
count=1;
% Determine how large the neighborhood is to be for this test
%nnum=2;
% patnum is used if you want to sequentially train patterns rather
% than select them at random from the training set
patnum=1;
% Set the number of interations
for j=1:numit
if (patnum==temp2) patnum=1; else patnum=patnum+1; end
count=count+1;
[nvec,est,wv]=regsom(trainset,startpts,nnum,0,numit);
startpts=nvec;
% Set how often you would like to see the plot in plotsee
plotsee=100;
if(((count/plotsee)-round(count/plotsee))==0)
plot(nvec(:,1),nvec(:,2),'+');
count
end
end
status3dsom=nvec;
```

```
function [ov,m,wv]=regsom(pvec,changevec,neighnum,testcode,numreps)
% Implementation of a 3-D Self Organizing Map for learning Camera-Theta
% uses proportional win rule  with slow learning rate based on number
% of trials and number of points, designed to match distribution near end
% testcode is 1 or 0 and used to access network if desired
% neighnum is the number of nearest neighbors used in rule
ov=changevec;
npl=length(pvec);
% initialize variables
na=length(changevec);
learnrate=npl/numreps;
% select a pattern newp from pvec at random
newp=pvec((ceil(rand*npl)),:);
% Use this line if you want to train using all patterns in order
% and omit the above line
%newp=pvec(pn,:);
% calculate distance of stored patterns from new
% pattern and find closest pattern based only on xyz distances
fixit=ones(na,1);
newpvec=(fixit*newp);
diffs=newpvec-changevec;
% Note: diffs should have and index equal to the space used with more
% complex data e.g. robot arm data this will change to diffs(:,1:3)
%dist=sqrt(sum((diffs(:,1:3).^2)'));
dist=sum((diffs(:,1:3).^2)');
[sortvals,indexx]=sort(dist);
% if network is only to estimate a value return it in m and stop
if (testcode==1) m=changevec(winnum,:); return; else m=0;  end
for j=1:neighnum
if (sortvals(j)==0) divterm=0; else divterm=sortvals(1)/sortvals(j); end
cval=ceil(rand*na);
changevec(cval,:)=changevec(indexx(j),:)+...
learnrate*(diffs(indexx(j),:))*divterm;
end
ov=changevec;
wv=sortvals(1,:);
```

```
function status3dsom=tcjsom3(trainset,startpts)
% Routine to test a SOM performance on a random 3D grid. It is also a demo
% of the adequacy of the network's alpha and r values for a training set
axis([-.1 1.1 -.1 1.1]);
% Use this block only if you want a random set of points and plot them
% Unless you want to change the internal grid dimensions use even square
% root number of points
clg;
% this ignores the startpts argument passed to the routine in order to
% build a new set
startpts=[];
% This part of the code lets you build a distributed random set of points
% for i=1:10 startpts(i,:)=[rand rand rand rand rand rand rand rand rand rand];
%end
% Use this block if you want the points to start at random in a center region
% Numpts should have an even cube root otherwise modify code
numpts=125;
for i=1:numpts
 startpts(i,1)=(.49+rand*.02);
 startpts(i,2)=(.49+rand*.02);
 startpts(i,3)=(.49+rand*.02);
 startpts(i,4)=(.49+rand*.02);
 startpts(i,5)=(.49+rand*.02);
 startpts(i,6)=(.49+rand*.02);
 startpts(i,7)=(.49+rand*.02);
 startpts(i,8)=(.49+rand*.02);
 startpts(i,9)=(.49+rand*.02);
 startpts(i,10)=(.49+rand*.02);
 end
temp=length(startpts);
temp2=length(trainset);
% Show the starting x-y plot to get some idea if ordering is occuring
plot(startpts(:,1),startpts(:,2));
% Set parameter values for alpha and the radius
count=1;
a=.08;
r=.2;
patnum=1;
% set the number of interations at each alpha and radius
for j=1:15000
if (patnum==temp2) patnum=1; else patnum=patnum+1; end
count=count+1;
a=1.0;
[nvec,est,wv]=cjsom3(trainset,startpts,r,a,0,patnum);
% Set ratio of decreases for learning rate and radius
%a=a*.99999;
% r=r*.99999;
startpts=nvec;
% startpts
% pause;
% Set how often you would like to see the plot in plotsee
plotsee=100;
if(((count/plotsee)-round(count/plotsee))==0)
plot(nvec(:,1),nvec(:,2),'+');
% plot(trainset(:,1),trainset(:,2),'*');
% Use these if you want to see how the radius or alpha rate is changing
count
a
end
end
%hold;
% Plot a comparison of the final results to the ideal results for 100 pts
% plot(trainset(:,1),trainset(:,2),'x')
status3dsom=nvec;
%hold off;
```

```
function [ov,m,wv]=cjsom3(pvec,changevec,bubble,lrnrate,testcode,pn)
% Implementation of a 3-D Self Organizing Map for learning Camera-Theta
%  relations. Uses mesh defined above, below, left, right, front,back for
% each point in changevec
% if testcode is 1 it returns the net's estimate of the winner other wise
% if 0 it treat's  pvec as a training pattern
% lrnrate learning rate (proportion of change of previous point values)
% ov = network's structure reconfiguration in response to the new pattern
% dist = vector of distances of patterns from input
% bubble controls the width of influece field of a neuon
ov=changevec;
% Put in a couple checks on the size and value of lrnrates
if lrnrate<0 lrnrate=-lrnrate; end
if lrnrate>1 lrnrate=1.0; end
npl=length(pvec);
% initialize variables
na=length(changevec);
% select a pattern newp from pvec at random
newp=pvec((ceil(rand*npl)),:);
% Use this line if you want to train using all patterns in order
% and omit the above line
%newp=pvec(pn,:);
% calculate distance of stored patterns from new
% pattern and find closest pattern based only on xyz distances
newpvec=(ones(na,1)*newp);
diffs=newpvec-changevec;
% Note: diffs should have and index equal to the space used with more
% complex data e.g. robot arm data this will change to diffs(:,1:3)
% dist=sqrt(sum((diffs(:,1:3).^2)'));
dist=sum((diffs(:,1:3).^2)');
[wv,winnum]=min(dist);
% if network is only to estimate a value return it in m and stop
if (testcode==1) m=changevec(winnum,:); return; else m=0;  end
% [lv,lnum]=max(dist);
% Use these lines if you want to change all points less than bubble each trial
%for j=1:na
%     if dist(j)<=bubble
%winnum=j;
% create links from points to their neighbors to generate a mesh
vecsize=floor((na)^(1/3));
% Note vector should have an even root or explicitly be put in somvectomat call
% change the following x,y and z dimensions accordingly
% refpos is the x,y,z reference coordinate around which neighbors are defined
refpos=somvectomat(winnum,[vecsize vecsize vecsize]);
% Define the 26 coordinates for the neighbors relative to refpos
% Define center neighborhood of a point
uc=[(refpos(1)) (refpos(2)) (refpos(3)+1)];
dc=[(refpos(1)) (refpos(2)) (refpos(3)-1)];
lc=[(refpos(1)-1) (refpos(2)) (refpos(3))];
rc=[(refpos(1)+1) (refpos(2)) refpos(3)];
ulc=[(refpos(1)-1) (refpos(2)) (refpos(3)+1)];
urc=[(refpos(1)+1) (refpos(2)) (refpos(3)+1)];
dlc=[(refpos(1)-1) (refpos(2)) (refpos(3)-1)];
drc=[(refpos(1)+1) (refpos(2)) (refpos(3)-1)];
% Define front neighborhood (relative to the center point)
uf=[(refpos(1)) (refpos(2)-1) (refpos(3)+1)];
df=[(refpos(1)) (refpos(2)-1) (refpos(3)-1)];
lf=[(refpos(1)-1) (refpos(2)-1) (refpos(3))];
rf=[(refpos(1)+1) (refpos(2)-1) refpos(3)];
ulf=[(refpos(1)-1) (refpos(2)-1) (refpos(3)+1)];
urf=[(refpos(1)+1) (refpos(2)-1) (refpos(3)+1)];
dlf=[(refpos(1)-1) (refpos(2)-1) (refpos(3)-1)];
drf=[(refpos(1)+1) (refpos(2)-1) (refpos(3)-1)];
cf=[(refpos(1))  (refpos(2)-1) (refpos(3))];
% Define back neighborhood (relative to center point
ub=[(refpos(1)) (refpos(2)+1) (refpos(3)+1)];
```

```
db=[(refpos(1)) (refpos(2)+1) (refpos(3)-1)];
lb=[(refpos(1)-1) (refpos(2)+1) (refpos(3))];
rb=[(refpos(1)+1) (refpos(2)+1) refpos(3)];
ulb=[(refpos(1)-1) (refpos(2)+1) (refpos(3)+1)];
urb=[(refpos(1)+1) (refpos(2)+1) (refpos(3)+1)];
dlb=[(refpos(1)-1) (refpos(2)+1) (refpos(3)-1)];
drb=[(refpos(1)+1) (refpos(2)+1) (refpos(3)-1)];
cb=[(refpos(1))  (refpos(2)+1) (refpos(3))];
% Find out which serial coordinate the neighbors have in startpts
ucvec=sommattovec(uc,[vecsize vecsize vecsize]);
lcvec=sommattovec(lc,[vecsize vecsize vecsize]);
dcvec=sommattovec(dc,[vecsize vecsize vecsize]);
rcvec=sommattovec(rc,[vecsize vecsize vecsize]);
ulcvec=sommattovec(ulc,[vecsize vecsize vecsize]);
urcvec=sommattovec(urc,[vecsize vecsize vecsize]);
dlcvec=sommattovec(dlc,[vecsize vecsize vecsize]);
drcvec=sommattovec(drc,[vecsize vecsize vecsize]);
% front block
ufvec=sommattovec(uf,[vecsize vecsize vecsize]);
dfvec=sommattovec(df,[vecsize vecsize vecsize]);
lfvec=sommattovec(lf,[vecsize vecsize vecsize]);
rfvec=sommattovec(rf,[vecsize vecsize vecsize]);
ulfvec=sommattovec(ulf,[vecsize vecsize vecsize]);
urfvec=sommattovec(urf,[vecsize vecsize vecsize]);
dlfvec=sommattovec(dlf,[vecsize vecsize vecsize]);
drfvec=sommattovec(drf,[vecsize vecsize vecsize]);
cfvec=sommattovec(cf,[vecsize vecsize vecsize]);
% back block
ubvec=sommattovec(ub,[vecsize vecsize vecsize]);
dbvec=sommattovec(db,[vecsize vecsize vecsize]);
lbvec=sommattovec(lb,[vecsize vecsize vecsize]);
rbvec=sommattovec(rb,[vecsize vecsize vecsize]);
ulbvec=sommattovec(ulb,[vecsize vecsize vecsize]);
urbvec=sommattovec(urb,[vecsize vecsize vecsize]);
dlbvec=sommattovec(dlb,[vecsize vecsize vecsize]);
drbvec=sommattovec(drb,[vecsize vecsize vecsize]);
cbvec=sommattovec(cb,[vecsize vecsize vecsize]);
% if they are legitimate indexes of the vector use them otherwise do not
% Correct changevec by moving neighborhood points toward the new pattern
% proportional to the nearest pattern's distance
if (ucvec>0)&(ucvec<=na)
changevec(ucvec,:)=changevec(ucvec,:)+...
(wv/dist(ucvec))*(lrnrate)*(diffs(ucvec,:)); end
if (lcvec>0)&(lcvec<=na)
changevec(lcvec,:)=changevec(lcvec,:)+...
(wv/dist(lcvec))*(lrnrate)*(diffs(lcvec,:)); end
% Unique treatement for winning center point
changevec(winnum,:)=changevec(winnum,:)+...
(lrnrate)*(diffs(winnum,:));
if (dcvec<0)&(dcvec>=na)
changevec(dcvec,:)=changevec(dcvec,:)+...
(wv/dist(dcvec))*(lrnrate)*(diffs(dcvec,:)); end
if (rcvec>0)&(rcvec<=na)
changevec(rcvec,:)=changevec(rcvec,:)+...
(wv/dist(rcvec))*(lrnrate)*(diffs(rcvec,:)); end
if (ulcvec>0)&(ulcvec<=na)
changevec(ulcvec,:)=changevec(ulcvec,:)+...
(wv/dist(ulcvec))*(lrnrate)*(diffs(ulcvec,:)); end
if (urcvec>0)&(urcvec<=na)
changevec(urcvec,:)=changevec(urcvec,:)+...
(wv/dist(urcvec))*(lrnrate)*(diffs(urcvec,:)); end
if (dlcvec>0)&(dlcvec<=na)
changevec(dlcvec,:)=changevec(dlcvec,:)+...
(wv/dist(dlcvec))*(lrnrate)*(diffs(dlcvec,:)); end
if (drcvec>0)&(drcvec<=na)
changevec(drcvec,:)=changevec(drcvec,:)+...
```

```
(wv/dist(drcvec))*(lrnrate)*(diffs(drcvec,:)); end
% begin front block
if (ufvec>0)&(ufvec<=na)
changevec(ufvec,:)=changevec(ufvec,:)+...
(wv/dist(ufvec))*(lrnrate)*(diffs(ufvec,:)); end
if (dfvec>0)&(dfvec<=na)
changevec(dfvec,:)=changevec(dfvec,:)+...
(wv/dist(dfvec))*(lrnrate)*(diffs(dfvec,:)); end
if (lfvec>0)&(lfvec<=na)
changevec(lfvec,:)=changevec(lfvec,:)+...
(wv/dist(lfvec))*(lrnrate)*(diffs(lfvec,:)); end
if (rfvec>0)&(rfvec<=na)
changevec(rfvec,:)=changevec(rfvec,:)+...
(wv/dist(rfvec))*(lrnrate)*(diffs(rfvec,:)); end
if (ulfvec>0)&(ulfvec<=na)
changevec(ulfvec,:)=changevec(ulfvec,:)+...
(wv/dist(ulfvec))*(lrnrate)*(diffs(ulfvec,:)); end
if (urfvec>0)&(urfvec<=na)
changevec(urfvec,:)=changevec(urfvec,:)+...
(wv/dist(urfvec))*(lrnrate)*(diffs(urfvec,:)); end
if (dlfvec>0)&(dlfvec<=na)
changevec(dlfvec,:)=changevec(dlfvec,:)+...
(wv/dist(dlfvec))*(lrnrate)*(diffs(dlfvec,:)); end
if (drfvec>0)&(drfvec<=na)
changevec(drfvec,:)=changevec(drfvec,:)+...
(wv/dist(drfvec))*(lrnrate)*(diffs(drfvec,:)); end
if (cfvec>0)&(cfvec<=na)
changevec(cfvec,:)=changevec(cfvec,:)+...
(wv/dist(cfvec))*(lrnrate)*(diffs(cfvec,:)); end
% Back block
if (ubvec>0)&(ubvec<=na)
changevec(ubvec,:)=changevec(ubvec,:)+...
(wv/dist(ubvec))*(lrnrate)*(diffs(ubvec,:)); end
if (dbvec>0)&(dbvec<=na)
changevec(dbvec,:)=changevec(dbvec,:)+...
(wv/dist(dbvec))*(lrnrate)*(diffs(dbvec,:)); end
if (lbvec>0)&(lbvec<=na)
changevec(lbvec,:)=changevec(lbvec,:)+...
(wv/dist(lbvec))*(lrnrate)*(diffs(lbvec,:)); end
if (rbvec>0)&(rbvec<=na)
changevec(rbvec,:)=changevec(rbvec,:)+...
(wv/dist(rbvec))*(lrnrate)*(diffs(rbvec,:)); end
if (ulbvec>0)&(ulbvec<=na)
changevec(ulbvec,:)=changevec(ulbvec,:)+...
(wv/dist(ulbvec))*(lrnrate)*(diffs(ulbvec,:)); end
if (urbvec>0)&(urbvec<=na)
changevec(urbvec,:)=changevec(urbvec,:)+...
(wv/dist(urbvec))*(lrnrate)*(diffs(urbvec,:)); end
if (dlbvec>0)&(dlbvec<=na)
changevec(dlbvec,:)=changevec(dlbvec,:)+...
(wv/dist(dlbvec))*(lrnrate)*(diffs(dlbvec,:)); end
if (drbvec>0)&(drbvec<=na)
changevec(drbvec,:)=changevec(drbvec,:)+...
(wv/dist(drbvec))*(lrnrate)*(diffs(drbvec,:)); end
if (cbvec>0)&(cbvec<=na)
changevec(cbvec,:)=changevec(cbvec,:)+...
(wv/dist(cbvec))*(lrnrate)*(diffs(cbvec,:)); end
% end
% end
ov=changevec;
```

```
function status3dsom=tpullsom(trainset,startpts, numit, netsize,nnum)
% Routine to test a SOM performance on a random 3D grid. It is also a demo
% of the Infolding Network using the proportional win learning rule
% and a stochastic point sampling logic
% numit is the number of learning trials desired
% trainset is the matrix of potential training patterns
% startpts is the set of points
% netsize is the number of points desired in startpts
% nnum is the size of the neighborhood for this network test
axis([-.1 1.1 -.1 1.1]);
clg;
% this code ignores the startpts argument passed to the routine in order to
% build a new set of a given number of points
startpts=[];
% This part of the code lets you build a distributed random set of points
for i=1:netsize startpts(i,:)=[rand rand rand rand rand rand rand rand rand rand];
end
% Use this block if you want the points to start at random in a center region
%numpts=netsize;
%for i=1:numpts
% startpts(i,1)=(.49+rand*.02);
% startpts(i,2)=(.49+rand*.02);
% startpts(i,3)=(.49+rand*.02);
% startpts(i,4)=(.49+rand*.02);
% startpts(i,5)=(.49+rand*.02);
% startpts(i,6)=(.49+rand*.02);
% startpts(i,7)=(.49+rand*.02);
% startpts(i,8)=(.49+rand*.02);
% startpts(i,9)=(.49+rand*.02);
% startpts(i,10)=(.49+rand*.02);
% end
temp=length(startpts);
temp2=length(trainset);
% Show the starting x-y plot to get some idea if ordering is occuring
plot(startpts(:,1),startpts(:,2));
count=1;
% Determine how large the neighborhood is to be for this test
%nnum=2;
% patnum is used if you want to sequentially train patterns rather
% than select them at random from the training set
patnum=1;
% Set the number of interations
for j=1:numit
if (patnum==temp2) patnum=1; else patnum=patnum+1; end
count=count+1;
[nvec,est,wv]=pullsom(trainset,startpts,nnum,0,patnum);
startpts=nvec;
% Set how often you would like to see the plot in plotsee
plotsee=100;
if(((count/plotsee)-round(count/plotsee))==0)
plot(nvec(:,1),nvec(:,2),'+');
count
end
end
status3dsom=nvec;
```

```
function [ov,m,wv]=pullsom(pvec,changevec,neighnum,testcode,pn)
% Implementation of a 3-D Self Organizing Map for learning Camera-Theta
% using and infolding rule taking random  points and folding into locations
% intermediate between new point and closest neighbors
% testcode is 1 or 0 and used to access network if desired
% neighnum is the number of nearest neighbors used in rule
% pn is
ov=changevec;
npl=length(pvec);
% initialize variables
na=length(changevec);
% select a pattern newp from pvec at random
newp=pvec((ceil(rand*npl)),:);
% Use this line if you want to train using all patterns in order
% and omit the above line
%newp=pvec(pn,:);
% calculate distance of stored patterns from new
% pattern and find closest pattern based only on xyz distances
fixit=ones(na,1);
newpvec=(fixit*newp);
diffs=newpvec-changevec;
% Note: diffs should have and index equal to the space used with more
% complex data e.g. robot arm data this will change to diffs(:,1:3)
%dist=sqrt(sum((diffs(:,1:3).^2)'));
dist=sum((diffs(:,1:3).^2)');
[sortvals,indexx]=sort(dist);
% if network is only to estimate a value return it in m and stop
if (testcode==1) m=changevec(winnum,:); return; else m=0;  end
for j=1:neighnum
if (sortvals(j)==0) divterm=0; else divterm=sortvals(1)/sortvals(j); end
cval=ceil(rand*na);
changevec(cval,:)=changevec(indexx(j),:)+...
(diffs(indexx(j),:))*divterm;
end
ov=changevec;
wv=sortvals(1,:);
```

```
function error=somnettest(testset,somset,numtests,numneigh)
% general k nearest neighbor rule version of somnettest.m
% numneigh is the number of nearest neighbors to be averaged
% Routine to test cjsom3d.m  by picking numtest random values from
% testset picking minimum from the file that
% cjsom3d.m produced to capture the mesh, getting joint angles out, calculating
% the  xyz for those angles and comparing with the
% exact xyz from testset, error is a matrix of each mean squared error of
% in cartesian space
b=length(testset);
e=length(somset);
for m=1:e fix1(m,1)=1; end
for i=1:numtests
i
index=ceil(rand * length(testset));
tval=testset(index,4:7);
newpvec=(fix1*tval);
diffs=newpvec-somset(:,4:7);
dist=sqrt(sum((diffs.^2)'));
[wv,winnum]=sort(dist);
winner=zeros(1,3);
for k=1:numneigh
winner=winner + somset(winnum(k),8:10);
end
winner=winner/numneigh;
xyzest=rob3dfeval(.6,.4,winner);
difxyz=testset(index,1:3)-xyzest;
error(i)=sqrt(sum(difxyz.^2));
end
```

```
function netstats(vector)
med=median(vector)
xbar=mean(vector)
deviation=std(vector)
maximum=max(vector)
minimum=min(vector)
plot(vector)
return
```