1

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

1

AFIT/GEO/ENG/88D-1

SPEECH RECOGNITION USING NEURAL NETS AND DYNAMIC TIME WARPING

THESIS

Gary Dean Barmore
Capt, USAF

AFIT/GEO/ENG/88D-1

Approved for public release; distribution unlimited

AFIT/GEO/ENG/88D-1

SPEECH RECOGNITION USING NEURAL NETS AND DYNAMIC TIME WARPING

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Gary Dean Barmore, B.S., B.S.E.E

Capt, USAF

December, 1988

Approved for public release; distribution unlimited

Preface

The purpose of this effort is to demonstrate the feasibility of using Kohonen neural nets in recognizing speech. A dual level system of two Kohonen neural nets accomplished this. The first net breaks an utterance into a sequence of phonemes, a trajectory. The second net recognizes those trajectories as distinct words.

In performing the study, I am deeply indebted to my faculty advisor, Dr. M. Kabrisky, for his suggestions, support, and most particularly, acting as a sounding board for ideas. I would also like to thank Capt S. Rogers for his assistance and suggestions.

Gary Dean Barmore

Table of Contents

Page

Page

List of Figures

Figure Page

Figure Page

List of Tables

AFIT/GEO/ENG/88D-1

Abstract

The purpose of this study is to demonstrate the feasibility of using Kohonen neural nets in speech recognition. This is done by combining a first level Kohonen net with a word recognition algorithm which is either dynamic time warping (DTW) or a second Kohonen net.

A digitized utterance is sliced and processed to obtain a sequence of 15 component vectors. Each component corresponds to the energy in a selected frequency range. An utterance of the digits zero through nine is used to train the first Kohonen net. After training, an utterance input to the net produces a trajectory through the net. Each point on the trajectory corresponds to a node and a particular sound.

These trajectories are input to a word recognition algorithm. The first of these, DTW, compares unknown utterances to template utterances. It is a computationally intense, mathematical algorithm, and it was used primarily to test the preprocessing and neural net training procedures. The second algorithm is a second Kohonen neural net. Digits are assigned to each node so that when an unknown trajectory is input to the second net, the node that "lights up" identifies the utterance.

Using DTW, 99% isolated and 93% connected speech recognition rates are achieved. With the second Kohonen net, isolated speech is recognized at up to 96%, depending upon the net format.

Recommendations for future effort include increasing the vocabulary, using multiple feature sets and nets to attempt speaker independent speech recognition, and substituting a backward propagation multi-layer perceptron net for word recognition.

SPEECH RECOGNITION USING NEURAL NETS AND DYNAMIC TIME WARPING

I. Introduction

As our society becomes more technically oriented, the ability to efficiently direct both machines and computers becomes essential. One very promising way of increasing that control is through voice direction of computer controlled devices. Not only could the worker use hands and feet to control conveyances, appliances, computers or weapons, he could also use voice.

However, in addition to simply expanding the number of methods by which devices are controlled, voice recognition permits simplification of the interface process. Remember the last time you tried to play a tape on your friend's new stereo? How long did it take you to find the ON/OFF button? Imagine simply saying, "Stereo, power on". Considering the vastly more complicated process of controlling a fighter aircraft, the anticipated benefits from voice control in a cockpit are immeasurably greater. Thus, in today's complicated weapons environment, the benefit to the military of a working voice recognition system is obvious.

Problem

Accordingly, the purpose of this effort is to demonstrate the feasibility of a speech recognition system for connected speech—a system which uses neural nets to simulate the activity of neurons and potentially take advantage of parallel processing techniques. The feature set used to characterize speech, the neural net training process, and the word recognition algorithms (which process the output from the neural net) are varied to obtain the best performance.

Background

Unfortunately, the current state of the art in voice recognition does not allow an efficient transfer of information between humans and machines. Speech recognizers limited to the speech of one person, with distinct pauses between each word, have recognition efficiencies around 98% (1:57)—in ideal or noise free environments (2:29–32). These recognizers fail when the speech becomes continuous (without distinct pauses) or is not spoken by the one person who trained the recognizer.

Attempts to have machines recognize continuous speech as well as speaker independent speech are limited to small vocabularies, require immense amounts of computation, and are relatively unsuccessful (1:56; 2:29; 3:74–76; 4). Speaker independent speech recognition, in particular, requires using multiple templates (5:199) ; i.e., training the recognizer with more than one voice, thus limiting the usable vocabularies because of the increased computation.

Attempts to use a form of parallel processing (or at least a serial simulation) to minimize the impact of computation time include Kohonen's work with neural nets (6:15; 7:18; 8:184; 9). However, his work is directed more towards producing a phonemic transcription of speech rather than actual whole word recognition.

Problems involved in using neural nets to recognize speech include choosing the feature set with which to train the net (the age old problem of traditional pattern recognition), finding a net with the optimum performance, and processing the net output into actual words and sentences.

Definitions

Connected Speech. Since the problem involves recognizing connected speech, a definition is essential. Connected speech is simply an utterance in which distinct pauses are not *intentionally* provided between words. When distinct pauses are included, the utterance is identified as isolated speech. In connected speech, words may be pronounced distinctly (that is, the pronunciation is not chopped or smeared) or they may be run together (where the words are chopped or smeared at the boundaries). Additionally, the speed with which words are spoken varies. Typically, when isolated speech is performed, words are spoken more slowly than during typical conversations. The difference in speeds becomes an important factor in properly processing the neural net output into recognized words.

Speaker Independence. While the same person generally pronounces the same words similarly time after time, different individuals have different speech organs and distinguishably different sounding voices. In addition, a person from the Midwest pronounces words differently than someone from the South. Such differences create significant problems in recognizing speaker independent speech. Accordingly, the speech recognition system described here is developed using speaker *dependent* utterances.

Feature Set. A feature set represents or characterizes utterances in such a way that they are distinguishable from other utterances. Usually, the feature set characterizes only a small slice of time, thus requiring a sequence of slices to represent the whole utterance. Hence, the speech waveform is broken up into distinct time slices (possibly overlapping) which are then transformed into vectors defined from the chosen feature set.

For example, in this research, the feature set is the narrow band frequency spectrum. The components of the vectors representing each time slice correspond to the energy in sequential frequency bands. Thus an appropriate transformation is either the fast Fourier transform (FFT) (10:272-285) or a fast Hartley transform (FHT) (11) with some postprocessing to reduce the number of frequency bands to a usable number.

Neural Net. After the feature set is chosen and the representative utterance is sliced up and transformed into vectors, it is time to train and use a neural net.

The neural net simply approximates or simulates a set of interconnected neurons. In the present work, a "Kohonen neural net" (6:15–18; 7:18–20; 8:182–185) is a simplified simulation of a single two dimensional layer of interconnected neurons. Each node in the two dimensional Kohonen net corresponds to a neuron and is associated with a set of weights. The weights correspond to the strength of the connection between the inputs and each node, and can alternatively be thought of as multi–dimensional vectors (in a hyperspace) which characterize the respective nodes.

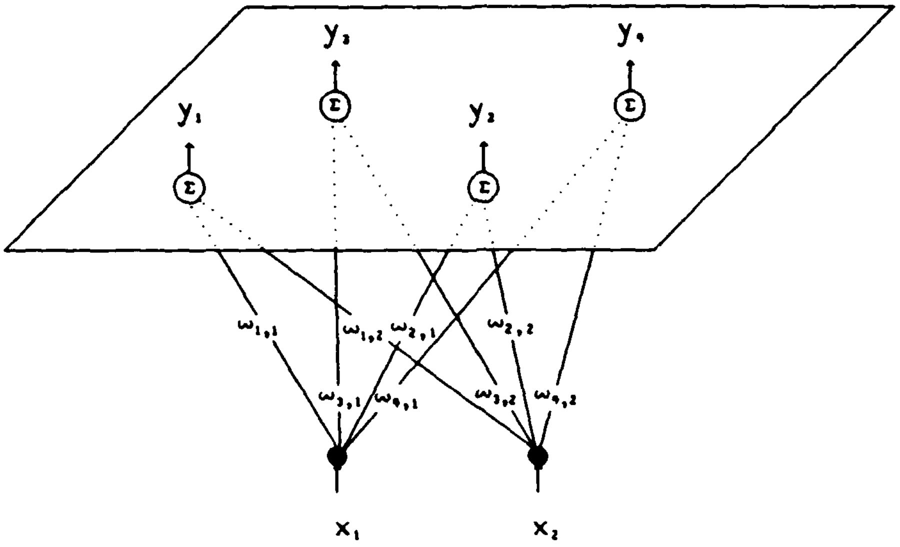
For example, if each input to the net consists of 15 components (such as a 15 component vector corresponding to a spectral slice of a speech waveform), each node is associated with 15 weights. The output of each node, corresponding to a given input, is the summation of the products of the weights to that node and the input components:

$$y_i = \sum_{j=1}^{N} \omega_{i,j} x_j \qquad (1.1)$$

where y_i is the output of node i, N is the number of input components, $\omega_{i,j}$ is the weight connecting the i^{th} node to the j^{th} component of the input, and x_j is the value of the j^{th} component of the input. For clarity, let the set of weights associated with a node be called that node's "weight vector". Figure 1.1 shows the relationship between the nodes, weights, and inputs in graphical form for a net having four nodes and two input components. The figure includes the equations describing a node's output as well as the weight update process (described in detail in Chapter II).

Training the net with a set of input vectors results in the weight vectors of the nodes approximating the input vectors. Also, the weight vectors associated with nodes in localized regions of the neural net tend to be similar to one another; i.e., the Euclidean distances between weight vectors, corresponding to a local region of the net, is relatively small.

Another way to think of a Kohonen net is as a codebook. If the input vectors representing time slices are thought of as phonemes, each node's weight vector represents a characteristic phoneme from the training set. Thus, when a phoneme is input to the net (as an input vector), the node



OUTPUT: $y_i = \Sigma \omega_{i,j} \; x_j$

TRAINING: $\omega_{i,j}(t+1) = \omega_{i,j}(t) + \alpha(t)\,(x_j(t) - \omega_{i,j}(t))$

Figure 1.1. A Sample Kohonen Neural Net. The first Kohonen net in this system actually has 225 nodes (in a 15 by 15 array) and 15 inputs.

(i) whose weight vector represents that phoneme will have the largest output (y_i). Obviously, each weight vector is just one of the codes from a codebook.

Trajectories. After a net is trained with an utterance representative of the vocabulary to be recognized (by repeating the utterance many times), the net is ready to characterize speech. Given an input, only one node of the net most closely resembles the input. Thus a sequence of time slices or vectors generate a trajectory through the neural net, with each point on the trajectory corresponding to the node most closely resembling the respective input vector.

Hence a neural net characterizes an utterance by a unique trajectory. It is these trajectories, whether they correspond to a single word, a string of spoken digits, or a sentence, which are processed to obtain the content of an utterance.

The trajectories may be thought of as either pictures of a complex meandering line or as sequences of two dimensional coordinates. The former representation suggests using traditional pattern recognition algorithms for processing images. The latter suggests traditional speech recognition processes such as dynamic time warping (DTW) (12:263–271; 13:Ch 3,6–16) or state transition mapping (14). This work examines the latter processes (DTW in particular) as well as the possibility of using a second neural net to process the Kohonen net output.

Dynamic Time Warping. A method is needed to evaluate the performance of different nets. For this purpose, Ney's one step dynamic time warping algorithm (12) is useful. This algorithm allows a near optimum recognition of connected speech while minimizing errors due to word boundaries and stretching of words during pronunciation.

Running a standard set of utterances through a net and then a dynamic time warping algorithm characterizes the performance of the net (including its training procedures and the feature set chosen for speech characterization) by the percentage of words correctly identified. Doing this for several nets (or feature sets) permits comparison of the nets (or feature sets) based on the variable being changed. And of course, this algorithm also provides a version of the last step in a speech recognition system, the word recognition algorithm.

Scope

The evaluation and selection of feature sets, the neural net training process, and the trajectory processing algorithm are all based upon a vocabulary of the spoken digits zero through nine. A set of standard utterances consisting of both single and connected digits is used to compare and optimize the various processes in the system.

Initial evaluations use the standard set of utterances and the one step dynamic time warping algorithm to evaluate both the feature sets and the neural net training process. Both DTW and a second Kohonen net are evaluated as word recognition algorithms.

Approach

In developing a system to recognize connected speech, the first task is to select a feature set. Once the feature set is selected, alternate training procedures for the Kohonen net are evaluated. After the net training process is tuned for optimum performance, by testing standard utterances with DTW, comparisons are made between likely algorithms to process the net trajectories and provide the sequence of words contained in the input utterances. Algorithms considered include variations of dynamic time warping (12; 14), state transition (15), phoneme analysis (16), reduced phoneme trajectories (17), fuzzy set analysis (18), and a second neural net. Time considerations limited tests to dynamic time warping and a second Kohonen neural net.

During each step, the evolving system is evaluated for its utility in recognizing both connected and isolated speech.

Sequence of Presentation

Chapter two presents the hardware and software environments used to digitize and process the utterances. Chapter three defines the speech recognition system and describes alternatives that were evaluated. Details are provided on each step and algorithm in the recognition process. Chapter four presents the results of the tests on those versions of the system which demonstrated reasonable performance. Results include tests for speaker dependent, isolated and connected speech. Chapter five provides conclusions and recommendations for further application and expansion of the recognition system. The Appendices contain additional data and the computer programs.

II. Development Environment

Every speech recognition system requires an analog to digital (A/D) convertor to sample the sound. This was provided by a Digisound Professional cartridge attached to an Atari 520 ST microcomputer. The sound data files were transferred to a VAX 11/780 where they were run through preprocessing and FFT routines.

All computer programs were developed on the VAX 11/780 in the C language under the VMS operating system. Most of the programs were run on either Micro-VAX II's or III's.

Sound Sampling

The Digisound Professional digitizer cartridge for the Atari 520 ST has a sampling range of 5 to 40 kHz. It takes eight bits (one byte) per sample in either a normal or logarithmic mode (where logarithmic mode allows a greater dynamic range than the normal mode). The data files generated by the Digisound software contain eight bytes of header information followed by N bytes of sound data, where N is the product of the number of samples per second times the number of seconds of data recorded. The format of each byte, assuming sound data has positive and negative values ranging about an average value translated to zero, has the negative values (−128 to −1) translated to the (+128 to +255) range by addition of 256.

Most of the data files generated for this work were sampled logarithmically at 16 kHz. Normal mode sampling was tested on a small amount of sound data and found to not significantly affect the performance of the system (at an early point in the system's development). The selection of 16 kHz and logarithmic sampling was arbitrary, but assured a "good" characterization of the sounds.

Software Development

All software was developed on a VAX 11/780 running the VMS operating system. Programs were developed exclusively in the "C" language using the Graphics Kernel System (GKS). The use of C and GKS permits porting of the system to other host computers with only minor software changes. Additionally, all sound files were stored on the VAX 11/780. This permitted one back-up tape to record all essential sound and program source files.

The programs were developed in a modular fashion. Generally, each step in the process was tested as it was developed. The use of *.com files (corresponding to the MAKE utility on UNIX machines) allowed modular compilation of routines and quick linking into executable modules.

Run-Time Environment

Most of the programs developed on the VAX 11/780 were run on either a Micro-VAX II or III. The exception to this was the transformation of sound files (*.snd) into files containing frequency domain vectors (*.trn) and dynamic time warping routines. These two sets of software were usually run real-time on the VAX 11/780 at about one million instructions per second (1 MIP).

All other software was usually run batch (on the Micro-VAX III at 3 MIP) or required GKS graphics (on the Micro-VAX II at .7 MIP). The software run on the Micro-VAX machines included training and testing of the neural nets.

Summary

A concise summation reveals the Atari 520 ST as the host for speech digitizing and VAX machines for all other work. Coding was done in C using GKS for graphics and VMS as the operating system. Finally, after identifying hardware and software used for development, its about time to describe the speech recognition system itself.

III. Speech Recognition System

The speech recognition system consists of a preprocessor, a first level Kohonen neural net, post-net processing, and a word recognition algorithm. The word recognition algorithm is either a dynamic time warping (DTW) algorithm or a second Kohonen neural net. The dynamic time warping algorithm works for either isolated or connected speech, while the second Kohonen neural net provides acceptable results only for isolated speech (with only limited *continuous* speech testing of the second Kohonen neural net). Figure 3.1 diagrams the steps in the system. Each step is described below in detail.

Preprocessing

This stage of the system was developed before the word recognition algorithm (see the section on Word Recognition Algorithms) and remained essentially unchanged after the initial development period. The preprocessing method was "frozen" when it created "reasonable" reduced trajectories (see the section on Post-Net Processing) through the first Kohonen neural net. Thus, it is possible that other, possibly simpler, preprocessing schemes will work as well, or even enable better overall system performance.

The preprocessing was initially very simple. The sound data, sampled at 16 kHz, was broken up into consecutive non-overlapping windows of 128 samples each. Each slice was run through an FFT routine to produce a set of 64 complex frequency coefficients. The amplitudes of these 64 frequencies were reduced to a set of 15 values corresponding to 15 frequency ranges. The extent of each range increased linearly as the frequency increased. Thus, for each time slice of 128 samples a 15 component vector, representing the frequency domain of the slice, was produced. The final preprocessing step was energy normalization of each vector to remove variations in the volume of the utterance.

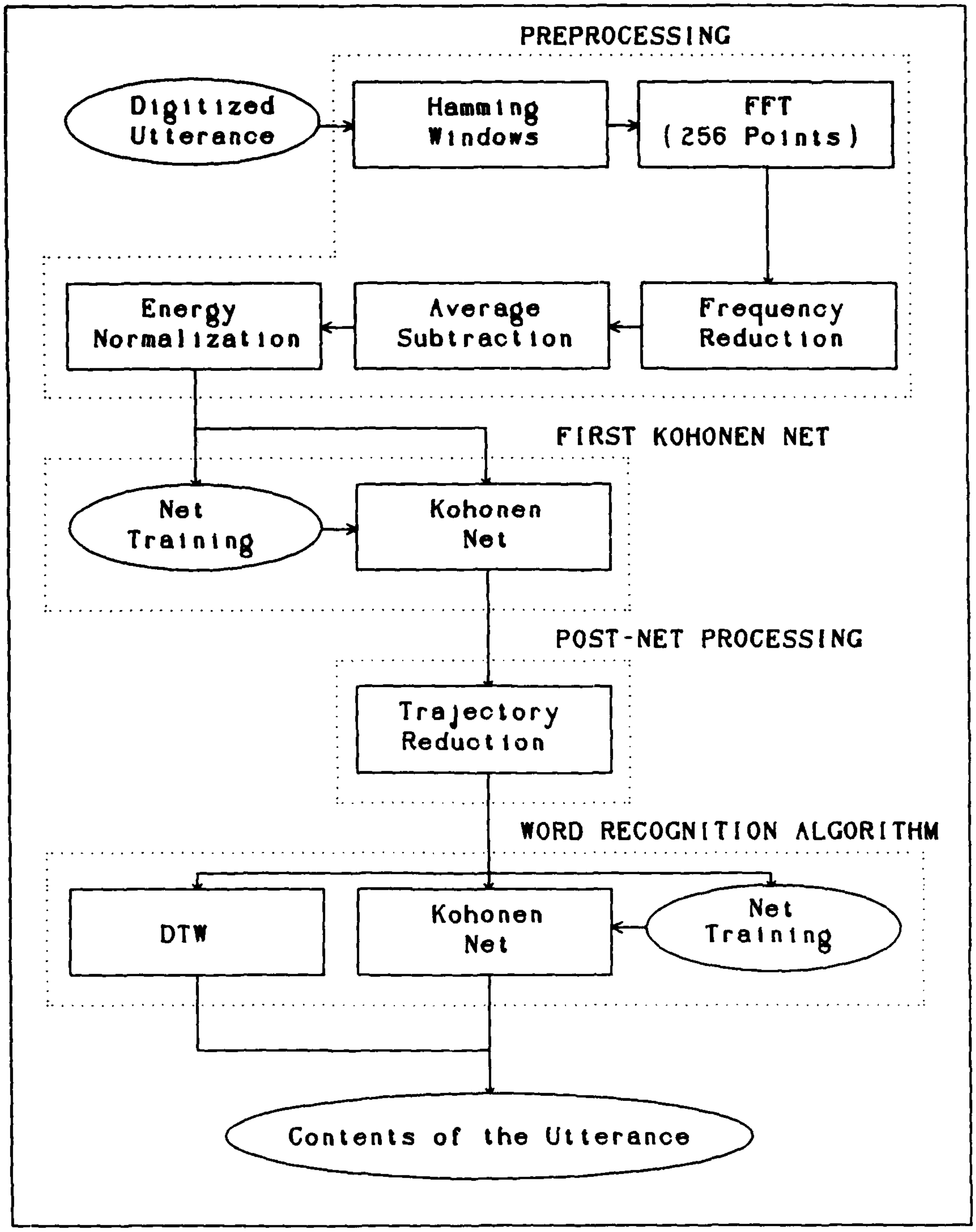


Figure 3.1. The Speech Recognition System. The second Kohonen net is developed and tested as an alternative to DTW as the word recognition algorithm.

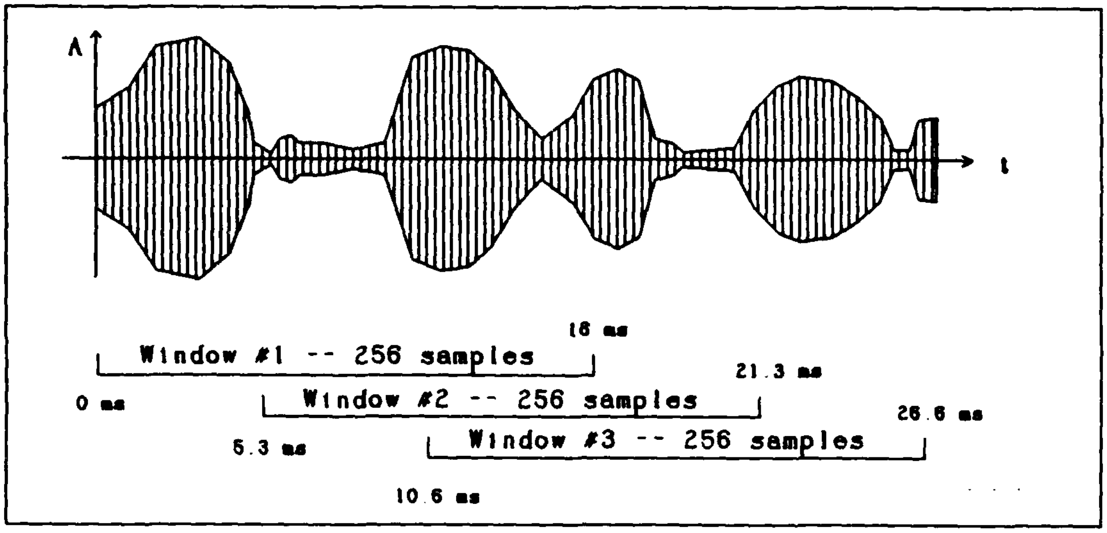


Figure 3.2. Hamming Windows. The ringing effects caused by rectangular windows are reduced by using hamming windows.

Unfortunately, this initial preprocessing scheme did not work. However, at this point, the preprocessing procedure was evaluated using untested first net training procedures and the full (not reduced) trajectories through the net. Accordingly, the following set of procedures was developed. Because of the reservations just listed, it is possible that this set provides more processing than is necessary.

Windowing. A hamming window reduces the high frequency ringing effects that would be caused by sampling speech with a rectangular window. And, since the period represented by each time slice is very short, the window length is increased to 256 samples. Thus, each hamming window covers 16 ms. An overlapping scheme of 3:1 is used to minimize boundary effects of the hamming windows. Thus, each 16 ms window begins 5.3 ms after the start of the prior window. The relation is depicted graphically in Figure 3.2. Please note that the sound envelope is shown compressed in time simply to assure that the viewer recognizes the data as that for speech.

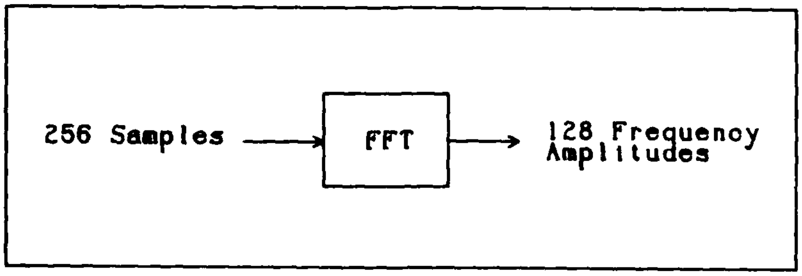


Figure 3.3. FFT Inputs and Outputs. The 128 complex coefficients are converted to positive amplitudes; the phase data are thrown away.

Fast Fourier Transform. A typical 256 point complex FFT routine processes each 256 sample hamming window. The imaginary components of the input are set to zero. The amplitudes of the resulting 128 frequency components are calculated from the real and imaginary outputs. This array of amplitudes is passed to the next preprocessing step. To complete the pictorial representation of preprocessing, Figure 3.3 is included.

Frequency Reduction. To reduce the time to obtain trajectories, the 128 amplitudes are reduced to 15. In speech, since there is less energy at the higher frequencies, a nonlinear compression of the frequencies provides some equalization of the spectral amplitudes (energies). In this work, a pseudo–logarithmic (nearly quadratic) reduction is used. In general, this allows a one–to–one transformation at the lower frequencies and simple summation of amplitudes from ever larger groups at the higher frequencies. It should be noted that the human auditory system uses an approximately logarithmic coding of frequencies.

This mapping was set up to provide a 15 component vector corresponding to each time slice (window) where each component *on the average* is of the same order of magnitude. Figure 3.4 provides some additional detail on the actual mapping.

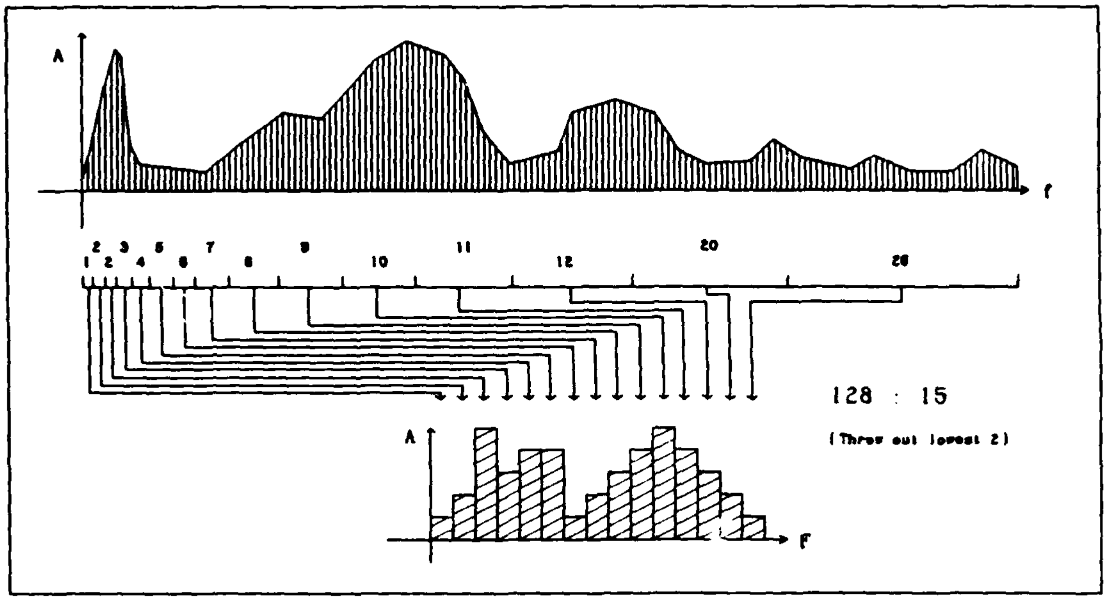


Figure 3.4. Frequency Reduction. After reduction, the sounds are still separable, but can be manipulated more quickly.

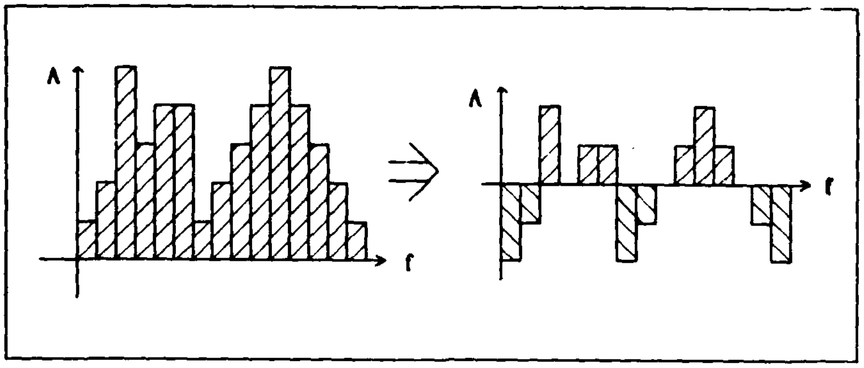


Figure 3.5. Average Subtraction. This process distributes the input vectors throughout a 15 dimensional hyperspace—not just the positive sector.

Average Subtraction. Since each slice is reduced to a 15 component vector, a sequence of slices can be thought of as a trajectory through a 15 dimensional hyperspace in which only the positive sector is used.[1] By finding the average value of the 15 components and then subtracting that average from each component, the components become both positive and negative, and all sectors of the hyperspace are used. This is shown in Figure 3.5.

Energy Normalization. Energy normalization of the 15 component vectors has two benefits. The first is that variations in the loudness of the speech will not create differences in the same sound's vectorial representation. Hence a word can be recognized no matter what its volume—assuming no other differences in pronunciation. Figure 3.6 shows the normalization process in terms of a unit hypersphere.

The second advantage of energy normalization is that neural net processing of normalized inputs allows use of dot products in place of Euclidean distance calculations (see Equation 1.1).

[1] Each frequency amplitude resulting from an FFT is defined as a positive value. And sums of these amplitudes are assigned to the 15 components of each vector. Thus all components of the vectors are positive. The positive sector in a hyperspace contains all of those vectors and is simply that region where all the coordinates of a point are positive.

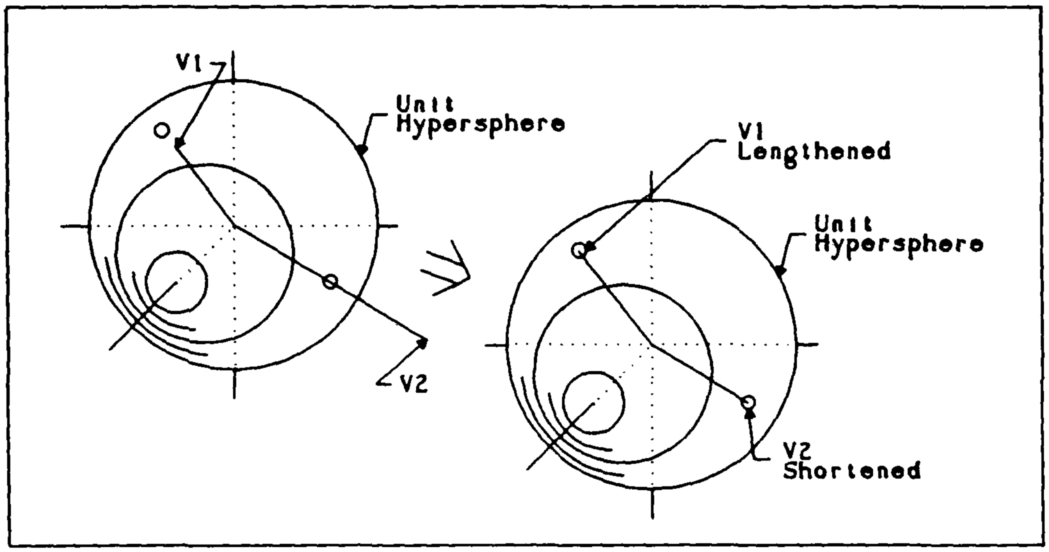


Figure 3.6. Energy Normalization. Variations in volume are removed when each slice is normalized to one.

This improves the speed of net training, testing, and use.

First Level Kohonen Neural Net

Preprocessing produces a file that consists of a sequence of normalized 15 component vectors. Each of these vectors, when applied to a Kohonen neural net, will "light up" a node. A node is defined as "lighting up" when the distance between the node's weight vector and the input vector is smaller than the distance between any other node's weight vector and the input vector. Hence, a sequence of input vectors (corresponding to the slices of an utterance) is represented by a trajectory of nodes lighting up.

Training. To get trajectories to use as templates for known words or as representations of unknown utterances, one must first have a trained net. The purpose of training the net is to get a two dimensional representation of the 15 dimensional space which produces: (a) very similar trajectories for different utterances of the same word, and (b) significantly different trajectories for utterances of different words. If these two requirements are not met, word recognition will not be successful.

Size. But what size net can meet these requirements? Since the vocabulary is the digits zero through nine, the number of significantly different sounds in those words should dictate the size of the net. Using an arbitrary set of phonemes (sounds), the ten digits were broken up into combinations of about 20 different phonemes. Assuming each region on the neural net corresponds to a phoneme, and assuming the reasonable size of a region is 9 nodes (to allow variations), the desired *rectangular* net has at least 180 nodes. This allows 9 versions of each of the 20 phonemes and theoretically accounts for most differences in pronouncing the same word. Thus, to allow margin for background noise in the training input data, the net was chosen to be 15 by 15 (a total of 225 nodes) with 15 weights per node.

Initialization. The weights for each node are initially assigned random values between -0.05 and $+0.05$. Since the inputs are normalized to unit energy, the average input component (corresponding to a weight) is $x_{i,ave} = \pm\sqrt{\frac{1}{15}} = \pm 0.258$.

Training Cycles. The actual training of the net requires applying an input (a 15 component vector) and updating the node's weights. This cycle is done repeatedly. The actual input vectors are chosen sequentially from an utterance which contains the spoken digits, in sequence, with brief pauses between words. When the number of training cycles is greater than the number of vectors (slices) in the training utterance, the training utterance vectors are simply applied again. The final nets were trained between 90,000 and 150,000 iterations (cycles). In analyses of early tests, most of the weight changes were accomplished within the first 10,000 iterations.

Random application of the vectors and random application of the words were also tried, but resulted in no improvement in recognition accuracy.

Neighborhoods. As previously described, when an input vector is applied, a node lights up. During any training cycle, ***only*** the weights of those nodes, in a neighborhood about the node that lit up, are updated. Of course, the neighborhood includes the node that lit up.

Neighborhoods are rectangular[2] in shape, and are truncated (where applicable) at the edge of the net. Neighborhoods are specified by their radius (or half their length and width). Thus a neighborhood specified as '4 3' actually includes an array of nodes sized 9 by 7. A typical training process on a 15 by 15 net running 90,000 iterations starts with neighborhoods sized '7 7' and linearly reduces them to '1 1' at iteration number 20,000. Thereafter, the neighborhoods stay at a constant '1 1'.

In the training processes used here, neighborhoods are never reduced to only the node which lit up. Since the minimum neighborhood includes the nearest neighbors, weights in complete regions (nine nodes) will be adjusted during all training cycles.

Gains. A gain curve specifies how much the weights of each node are changed during each training cycle (within the specified neighborhood). Outside of the neighborhood, node weights are not changed. Typical gain curves are piecewise linear in two sections. For example, the gain might start at 0.1 and reduce linearly to 0.0 at 20,000 iterations. Thereafter it might reset to 0.01 at 20,001 iterations and reduce linearly to 0.00 at 90,000 iterations.

The actual equation used to update a weight within the given neighborhood is:

$$\omega_{i,j}(t+1) = \omega_{i,j}(t) + \alpha(t)(x_j(t) - \omega_{i,j}(t)) \tag{3.1}$$

where ω is the weight connecting node i to the j^{th} component of the input x, t is time (or cycle), and α is the gain.

Conscience. While training a Kohonen neural net as described above, one usually finds that several nodes in the net (during the training cycle) are not often lit up by the inputs.

[2] Although the software allows neighborhoods to be rectangular, all of the nets in this effort were trained using square neighborhoods.

This results in under-training of certain nodes and a corresponding under-utilization of the same nodes during later use of the net. Specifically, the possible set of trajectories representing the chosen vocabulary will never, or seldom, include some of the nodes.

This is a problem only if different classes (regions in the 15 dimension hyperspace representing different sounds) tend to be pushed together in a region of the net, not allowing sufficient resolution for adequate speech recognition.

One possible cure for such a problem is applying "conscience". The name is derived (rumor has it) from the assumption that given a large enough set of random inputs, any node *should* light up with equal probability. Supposedly, the net training routine is "bugged" when this does not occur and thus activates its "conscience".

The version of conscience used here is simple. It assumes that during training, the inputs should have lit up each of the nodes about an equal number of times at every cycle of the training process. Thus, when the program is searching through the 225 nodes to find the one that lights up (for each cycle), it does not consider those nodes which have already been lit up too often. The routine specifically uses the following equation for eligibility of the node:

$$c_i(t) < \beta\frac{t}{n} \tag{3.2}$$

where c is the number of times node i has been lit up prior to time (cycle) t, n is the total number of nodes, and β is a conscience factor (usually 1.5).[3] If the conscience factor is changed to any number much greater than 1.5, the program effectively implements *no* conscience. If Equation 3.1 is true for a given node, it is eligible for consideration as the closest node to that input.

Use. After the net is trained, it is ready for use. Using a net is the same whether the desired output is the trajectory for a template (a known utterance) or an unknown utterance. A sequence of 15 component vectors, prepared by preprocessing, is presented one at a time as input to the

[3] Note that even if a node is not eligible as the closest node to a given input, its weight vector will still be updated if it is within the neighborhood of the node that did light up. When a node's weight vector is updated this way, the value of c_i is *not* incremented.

net. The resulting sequence of nodes that light up is the output—a trajectory. The length of the trajectory equals the number of slices (windows) generated in preprocessing.

Depending on the use of the output, the nodes are identified either as an x-y coordinate pair or as a scalar. When the x-y representation is used, the range of both x and y is 0 to 14. When the nodes are represented as scalars they range from 0 to 224 with 0 to 14 assigned to the first row of nodes and 210 to 224 assigned to the last row.

Post-Net Processing

After a trajectory is generated it can be input to a word recognition algorithm such as DTW or a second Kohonen neural net. However, examination of typical trajectories suggests that some post-net processing be applied prior to using a trajectory as input to a word recognition algorithm.

Rationale. One expects the trajectory to stop at certain places on the net for periods of time. Such pauses in the trajectory should correspond to the pronunciations of distinct vowel phonemes[4] within each word. Of course, because of the dynamic nature of human speech, one would expect a given phoneme to wander within a region of the net (assuming the net trained each phoneme to a specific region). And if the net trained any given phoneme to more than one region of the net, a trajectory might pause either on multiple, non-adjacent regions during a phoneme's pronunciation or on different regions during different utterances of the same word. But in any case, the expected trajectory is a sequence of pauses at certain locations with transitory jumps between those pauses. This is observed. Since the transitory jumps between pauses are inconsistent between different utterances of the same word, one expects their elimination to improve accuracy.

Method. The actual implementation requires two passes through a stored trajectory. If any point on the trajectory is not within two node units (using the x-y coordinate pair representation) of another point on the trajectory within two time slices in either direction, it is eliminated. In the

[4] Even if the phonemes are arbitrarily defined.

second pass, all points on the trajectory which are not part of three consecutive points, each point within two node units of the others, are eliminated.

This procedure leaves only those points on the trajectory that pause in a given phoneme region for at least 26.6 ms (three overlapping time slices). Phoneme regions sized differently than two node units were not tested. This processed trajectory is called a "reduced trajectory".

Tests were run on both the full trajectory, the reduced trajectory, and a reduced trajectory with consecutive pauses at any particular node reduced to one point on the trajectory. The second form (the reduced trajectory) performed best.[5]

Word Recognition Algorithms

Two generic types of word recognition algorithms were developed. The first was an implementation of Ney's one pass dynamic time warping algorithm (DTW) (12). This DTW algorithm permits recognition of either isolated or connected digits. Unfortunately, it is rather time consuming. Thus, a second Kohonen net was also developed to recognize words. The second Kohonen net can recognize isolated words, but does not approach the performance of DTW in recognizing continuous speech.

Dynamic Time Warping. Ney's one pass dynamic time warping (12) was adapted almost unchanged using the output of the first Kohonen net. The significant changes are the much simpler inputs (trajectories), testing of the "stretch" factors, and adapting the routine to test a set of standard utterances automatically.

Description. Although significant memory can be saved by coding shortcuts, it is easiest to describe the algorithm by thinking of one large array. Figure 3.7 shows a simplified view of a completed array. Assign to each column one vector or time slice of the utterance under test. In the process here, that vector is an x–y coordinate pair corresponding to a point on the reduced

[5] See Figures 4.3 through 4.5 for examples of full and reduced trajectories and the reduction process.

trajectory. The first column is assigned the first vector, the last column the last vector. To each row is assigned a vector from one of the 11 templates. The templates are typical utterances of the ten digits, one each, and a short period of background noise (silence). Thus, the first vector of the digit zero is assigned to the bottom row and the last vector of the trajectory for silence is assigned to the top row. Silence is included since it is part of any recording of natural speech. Using a template for silence alleviates the need for trimming such periods from a digitized recording.

To explain DTW and how the array is processed, first assume a *simplified* array with only one template (e.g., a two) and an utterance to be identified which also happens to be a two. Vectors from the two trajectories are assigned to the rows and columns of an array. The size of the array obviously depends on the lengths of the trajectories. Fill in the array by assigning to each element the distance between the vectors assigned to that element's row and column. Now, if both words were spoken identically, the diagonal would be filled with zeros.

Of course, the same word, spoken twice by the same person can be either longer or shorter, and generally is stretched non-uniformly throughout the utterance. The DTW algorithm tries to find the best possible match between two words despite such stretching. To do so, the algorithm starts at the bottom left element of the distance array (just generated) and finds the shortest cumulative path to the upper right element. Any path can go diagonally (the theoretical best match), right one element (utterance stretched more than template), or up one element (template stretched more than the utterance). Factors are assigned to each of these directions which show preference for the direction and amount of stretch expected by the programmer. Figure 3.8 demonstrates this process with two short, hypothetical twos.

Thus, the cumulative minimum path, V, is assigned a value according to the following equations:

$$V = \min(\delta)_i = \min(\sum \gamma_j \, d_{r,c})_i \tag{3.3}$$

where V represents the best match between the two trajectories, i represents all possible paths

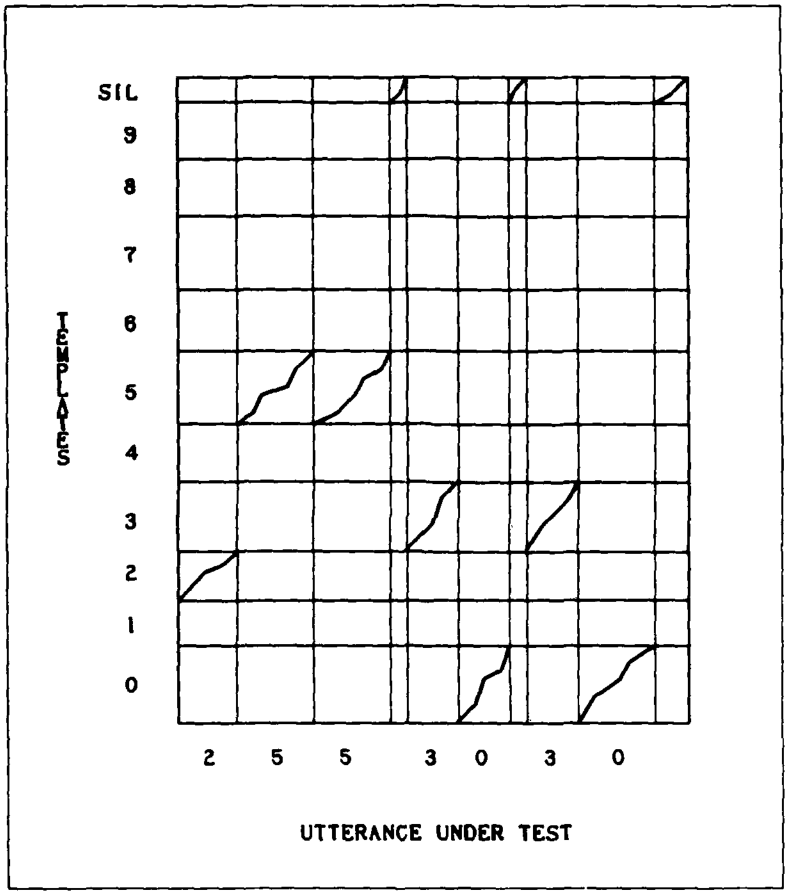
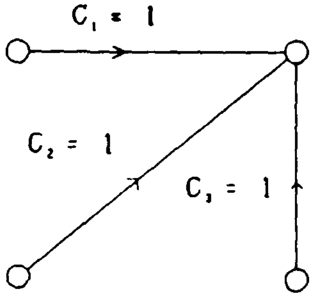


Figure 3.7. DTW Array. The problems caused by stretching and word boundaries are handled efficiently in a one-pass dynamic time warping algorithm.

(1) Distance Array

	(9.8)	15	7	**2**
Template Two	(4.6)	8	**0**	7
	(4.5)	7	**1**	8
	(0.1)	**1**	9	16
		(1.1)	(4.6)	(8.9)
			Test Two	

(2) Stretch Factors



(3) Cumulative Minimum Paths

	(9.8)	31	9	**4**
Template Two	(4.6)	16	2	9
	(4.5)	8	2	10
	(0.1)	**1**	10	26
		(1.1)	(4.6)	(8.9)
			Test Two	

Note: The cumulative minimum path is in bold characters. V equals 4.

Figure 3.8. A Simplified DTW Example. This simple problem shows how the basic routine works when the utterance under test is compared to only one template.

through the distance array, γ_j is a coefficient depending on which of the three possible directions, j, is chosen (usually 1, 0.75, and 0.75 for diagonal, right, and up), and $d_{r,c}$ is the distance between vectors assigned to row r and column c. If values of V are found for a given unknown utterance when compared to each of the 11 templates (in 11 arrays), the unknown utterance is probably the word represented by the template with the smallest V.

In the simple case where one unknown word is compared to one template (doing this eleven times), the *coding* to find V is simple. Just start at the bottom of the left column of the distance array and proceed upwards–column by column. As each element in the array is reached, a cumulative minimum partial path value is assigned to the same element of a different array (of the same size). The partial path value is found by adding the distance assigned to that element (multiplied by the appropriate stretch factor) to the minimum value chosen from the partial values calculated for the elements to the left, diagonal, or down directions. Thus, the value of V is just the cumulative minimum partial path value of the top right element.

When the unknown utterance contains connected digits, the process is slightly more complicated and the array of Figure 3.7 must be used. The same column by column calculations are made (proceeding upward through each column), but special rules take effect as template boundaries are crossed. When a boundary is reached, one assumes either a word just ended or else stretch is occurring in the existing word. Thus, at the first row after each template boundary, the minimum partial value is chosen from the elements corresponding to the last row of each template (words ended) in the prior column, and the element just to the left of the one under consideration (stretching of the word). Finally, one *must* also assume that the end of the trajectory for the utterance under test corresponds to the completion of a digit (or silence).

To find the contents of an utterance, the minimum path through the array is traced from the upper right element to the lower left, tracing out the various templates as the path progresses. As mentioned earlier, there are coding schemes which do not require saving more than two rows of pointers and two columns of partial path values at any one time.[6]

Stretch Factors. The stretch factors assigned to the three path directions seem to have a large impact on the success of recognizing both isolated and connected digits. The greatest success was found when factors of 1.0, 0.75, and 0.75 were assigned to the diagonal, right, and up directions, respectively. This differs from Ney's suggested values of 1.0, 2.0, and 0.5. The best values actually depend on the specific utterances used.

Automatic Scoring. As previously mentioned, a set of standard utterances and templates are used to evaluate the success of particular versions of the system. To minimize the operator's efforts, an automatic scoring routine was developed.

A scoring DTW array is used where sequences of digits are assigned to the rows and columns. For example, if the utterance is known to be the words 1-2-3, those integer values are assigned to the three *rows* of the DTW array. After running through the DTW process described earlier, the utterance might be found to be 1-2-4. Thus, 1, 2, and 4 are assigned to the three *columns* of the scoring DTW array. The distances assigned to each element are either zero or one. Zero is chosen if the row and column match; one if they do not. Figure 3.9 shows this example.

The stretch coefficients are set to 1.0, 0.5, and 1.0 for diagonal, right, and up directions. In the scoring DTW array, these correspond to a match that is a substitution (or correct), an insertion, or a deletion. Therefore, when V is calculated it becomes the number of errors. Knowing the number of actual digits in the utterance allows calculation of the percentage correct.

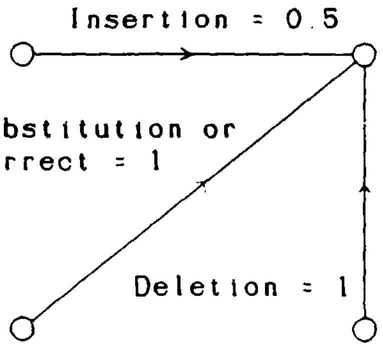
Second Kohonen Neural Net. Obviously, the computation time needed to use a DTW algorithm increases in direct proportion to the size of the vocabulary (number of templates) to recognize. In an attempt to reduce computation time, a second Kohonen neural net was evaluated as an alternative to the DTW algorithm.

[6] See the routine "cdtw" in the program "autodtw" in Appendix B, page B-48.

(1) Distance Array

Template			
3	1	1	1
2	1	0	1
1	0	1	1
Test	1	2	4

(2) Stretch Factors



(3) Cumulative Minimum Paths

Template			
3	2	1	1
2	1	0	0.5
1	0	0 5	1
Test	1	2	4

Note: The cumulative minimum path to the upper right element is the number of errors

Figure 3.9. DTW Used for Scoring. This example shows how DTW is adapted to automatically score the results of a test of connected speech recognition.

Inputs. Because DTW works well on reduced trajectories, the "reduced trajectory" contains enough information to identify words. But trajectories vary in length and cannot be energy normalized without destroying the information.

Thus, the first form of input considered was simply a set of 225 values corresponding to the 225 nodes of the first net. Each value is the number of times the trajectory hit that node. The second idea is not as simple, but includes more information in the input vector. When a slice (vector) is input to the first net, it produces an activation surface identified by 225 values (the dot product of the input vector with the node's weight vector for each node). Summing[7] these surfaces produces a composite surface which might have local maxima where the trajectory paused longest. Both forms of input can be normalized. However, they inherently *do not* contain the time dependent information in the trajectories. Success was not achieved using either of these techniques.

Alternate forms of input vectors which retain the time dependency are either a set of 200 values corresponding to 100 x-y coordinate pairs or 100 values corresponding to 100 scalars. Both sets of values are simply "reduced trajectories" through the first net. The vast majority of isolated digits produce reduced trajectories of less than 100 points. When the trajectory lengths are less than 100 points, both of the alternate forms of input vectors are filled with -1's to maintain a constant length input. Both of the alternate forms were reasonably successful.

Size. The desired output is either one of the 10 digits or silence; that is, eleven words. Assuming some variation in the words, regions of nine nodes should adequately represent each word. Thus a net of 10 x 10 nodes provides more than the 99 nodes desired. The number of weights in a node's weight vector is either 100 or 200 depending on the form of input selected (scalar or x-y pair).

[7] Summing two surfaces simply adds the height at any point on one surface to the height at the corresponding point on the second surface.

Training. Training is very similar to that of the first net. The biggest difference is that here the number of training input vectors is the same as the number of nodes. In the first net, the number of inputs was almost an order of magnitude larger than the number of nodes.

Additionally, initial weights are randomly assigned from sets consisting of the integers from 0 to 225 (where the inputs are 100 scalars) or 0 to 14 (where the inputs are 100 x-y coordinate pairs).[8]

Traditional Training. Two sets of training inputs were selected and used for various nets. The first of these was a set of 100 words, 10 each of the 10 digits. The second was a set of 91 words, 9 each of the 10 digits and 1 example of silence.

Training runs included cases for no conscience as well as conscience factors of 1.1 and 1.5. To account for the decreased eligibility of nodes during training with conscience, the training process was increased to 150,000 cycles.

Upon completion of each net's training, the set of training inputs was applied to the *trained* net. With no conscience, only about 58% of the nodes light up. This implies a need for conscience in training the second net.

Finally, there is an obvious question implied by training a net with 100 *scalar* inputs. The nodes are made to reflect the inputs by pulling their weights towards the input components. But two points on a trajectory can be adjacent nodes on the first net while being 15 units apart on the scalar input representation. Thus, can the existing training process be effective in overcoming those 15 unit discontinuities?

Abbreviated Training. Clearly, using a second net is little more than selecting a set of codebook trajectories. As such, does one expect training to generate 100 codebook trajectories that are better than an arbitrary 100 original trajectories (10 each of the 10 digits)?

[8] Note that when scalar inputs are used, both the scalar input components and the initial node weights are scaled by $\frac{1}{226}$ to keep the values in the range of zero to one.

To answer this question, nets were generated for each of the two forms of inputs (100 scalars and 100 x-y coordinate pairs) without any training. In these nets, the reduced trajectories for ten examples of each digit were assigned directly to the weight vectors of each row of nodes. No other training was provided.

Use In Isolated Speech. Once a net is trained, a digit is assigned to each node. Thereafter, when a reduced trajectory is applied to the input of the net, the net identifies the input as the digit assigned to the node that lit up.

Digits are assigned to nodes by comparing the weight vector of a node to the trajectory representations of a set of 100 digits (ten each) ***not used*** in training the net. The digit whose trajectory is "closest" to the node wins. Finding the "closest" trajectory involves either Euclidean distance or performing a mini-DTW.

Euclidean Distance. In this case, the distances are simply the sum of the squares of the differences between the weights and input components. At times, a TAXI distance is used rather than Euclidean. In TAXI space, distance is the sum of the absolute values of the differences between the weights and the input components

Dynamic Time Warping. When DTW is used, the 100 scalar trajectories are converted to 100 x-y coordinate pairs. The lengths of the utterance and template trajectories are always adjusted to eliminate the portion of the trajectories filled with trailing -1's. Use of DTW to find the closest digit considers the variations in stretch inherent in any utterance.

Use In Connected Speech. To use the second Kohonen net to identify connected speech, the same training and node assignment procedures are used. However, the methods tested to identify connected speech used ***only*** DTW to identify the closest node.

Each slice of the utterance under test is assigned a digit and a distance. The slice in question and the following 99 in the sequence are used to generate a trajectory through the first Kohonen net. The trajectory is then used to find which node lights up in the second Kohonen net. The comparison method here is DTW where the template and utterance (trajectory) lengths are both assigned the number of weights in the appropriate node's weight vector which are not -1's. The digit assigned to the node which lit up is the digit assigned to the slice. That slice is also assigned a weight or distance which is merely V (the cumulative minimum path distance) normalized by dividing it by the number of array elements in the minimum cumulative path.

Thus, when a sequence of distances assigned to a sequence of slices (i.e. an utterance) has an obvious local minima (one that lasts for more than about 50 ms and is at least 100 ms from another minima), it is assumed that the utterance begins a new digit at that point. In particular, it begins the digit assigned to that slice. In turn, the sequence of digits assigned to the sequence of local minima is interpreted as the content of the utterance under test.

Summary

This chapter has described in detail the speech recognition system developed in this effort. The description included preprocessing of the digitized speech, training of the first Kohonen net for production of trajectories, processing those full trajectories to eliminate transitory points, and evaluating the reduced trajectories by either a DTW algorithm or a second Kohonen neural net to obtain the content of an utterance. At no time was any attempt made to quantify the performance of the system[9]—that data was saved for the next chapter.

[9] Although at times it was necessary to verify some hypothesis or state that some procedure did or did not work.

IV. Results and Discussion

First Kohonen Neural Net

An ideally trained Kohonen net should place inputs from a given class within the same region (group of nodes). Is it possible to tell whether this result is obtained just by looking at a net? To some extent, yes. Figure 4.1 displays the net "speak1". Each of the 225 nodes is represented by its weight vector, shown as a small spectrum. The spectrum is simply the 15 weights drawn as vertical bars. It is easy to see that various regions of the net have similar spectra, and the changes between adjacent nodes are very graceful.

"Speak1" was trained with no conscience. Figure 4.2, "speak10", was trained with a conscience factor of 1.5. Again, one observes that within a region, the node spectra are very similar. In fact, without prior knowledge, there is no way visually to tell that conscience was used in training "speak10" and not in "speak1". This is expected. It shows that training has occurred, but does not show whether a net will be successful in producing identifiable trajectories.

Trajectories

Trajectories can be viewed either graphically, or as a sequence of integers representing the 225 nodes. Figure 4.3 graphs a full trajectory for the word zero. Figure 4.4 shows the reduced trajectory of the same word. Figure 4.5 shows the reduction process from a full trajectory to a reduced trajectory as a printout of integers (each integer one of the 225 nodes). The graphs contain the number of the slice which lights up a node in the rectangle representing that node. If a node is lit up more than once, only the last slice number is identified and three asterisks are added to the rectangle. Note that slice numbers are only "effective" values for the reduced trajectories.

Appendix A shows the reduced trajectories for the templates (the digits zero through nine and silence) used in the DTW algorithm. When comparing the eleven templates they appear separable—even to the eye.

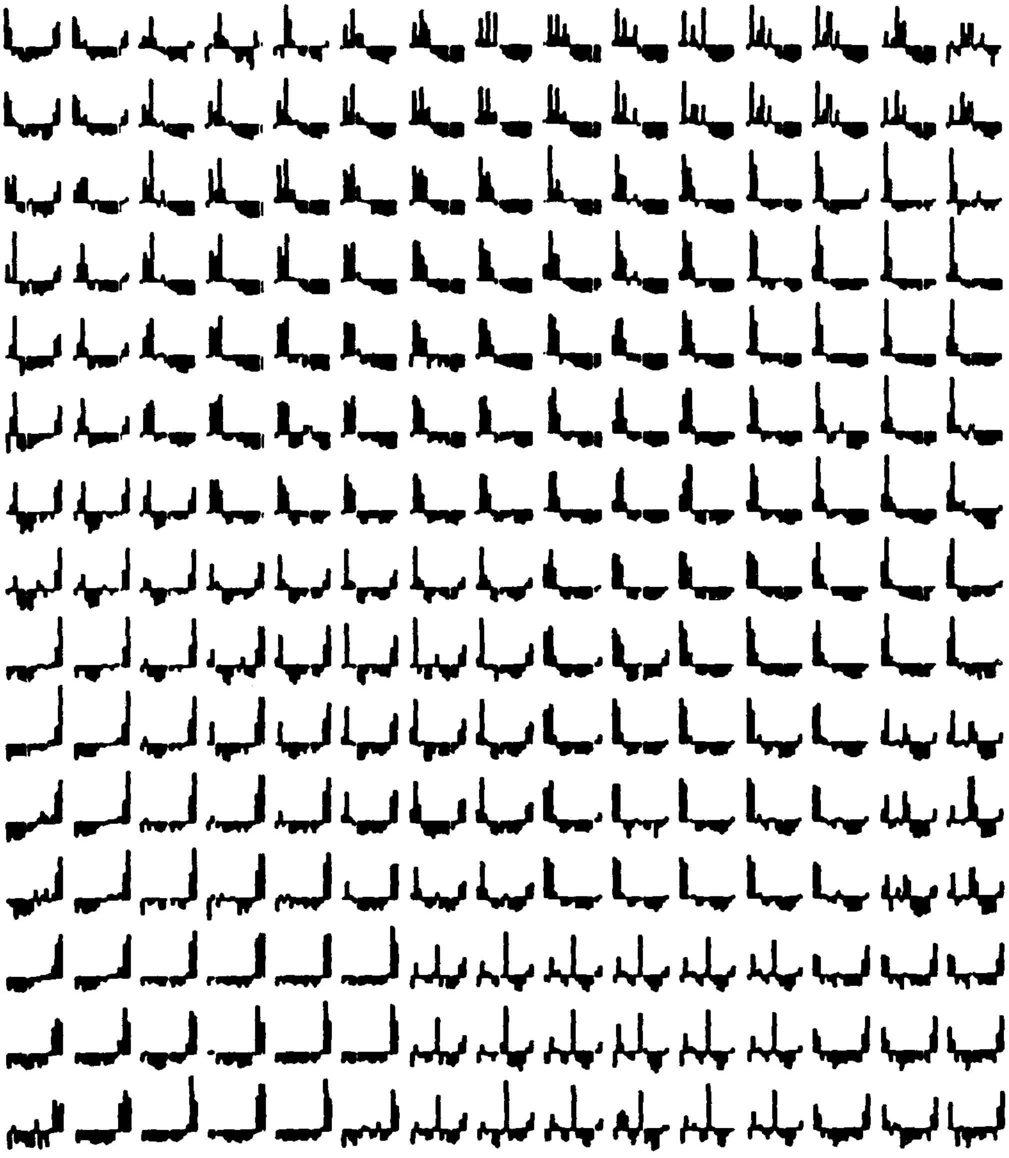


Figure 4.1. Net Speak1 Trained with No Conscience. Note the regional similarities and the gradual change from node to node and region to region.

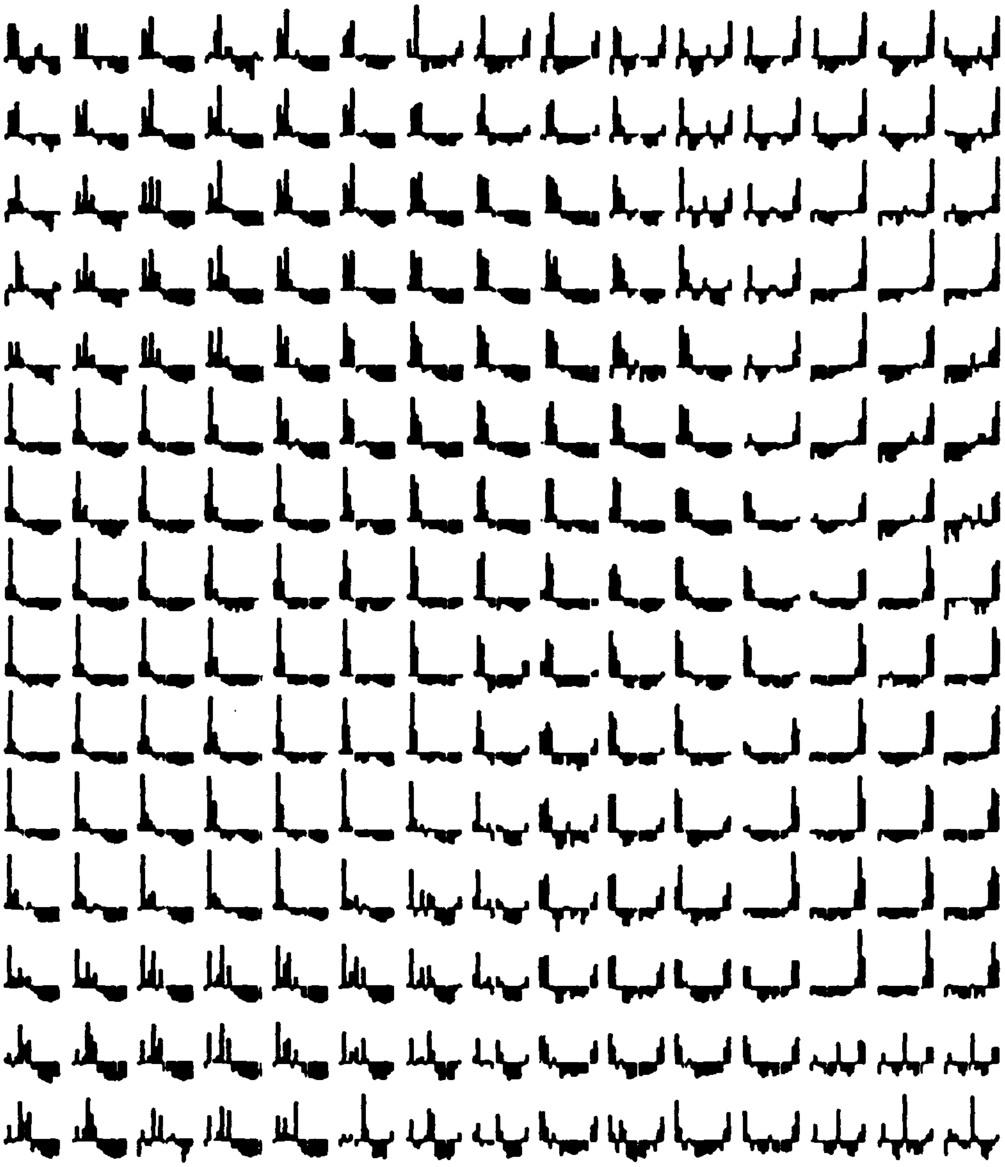
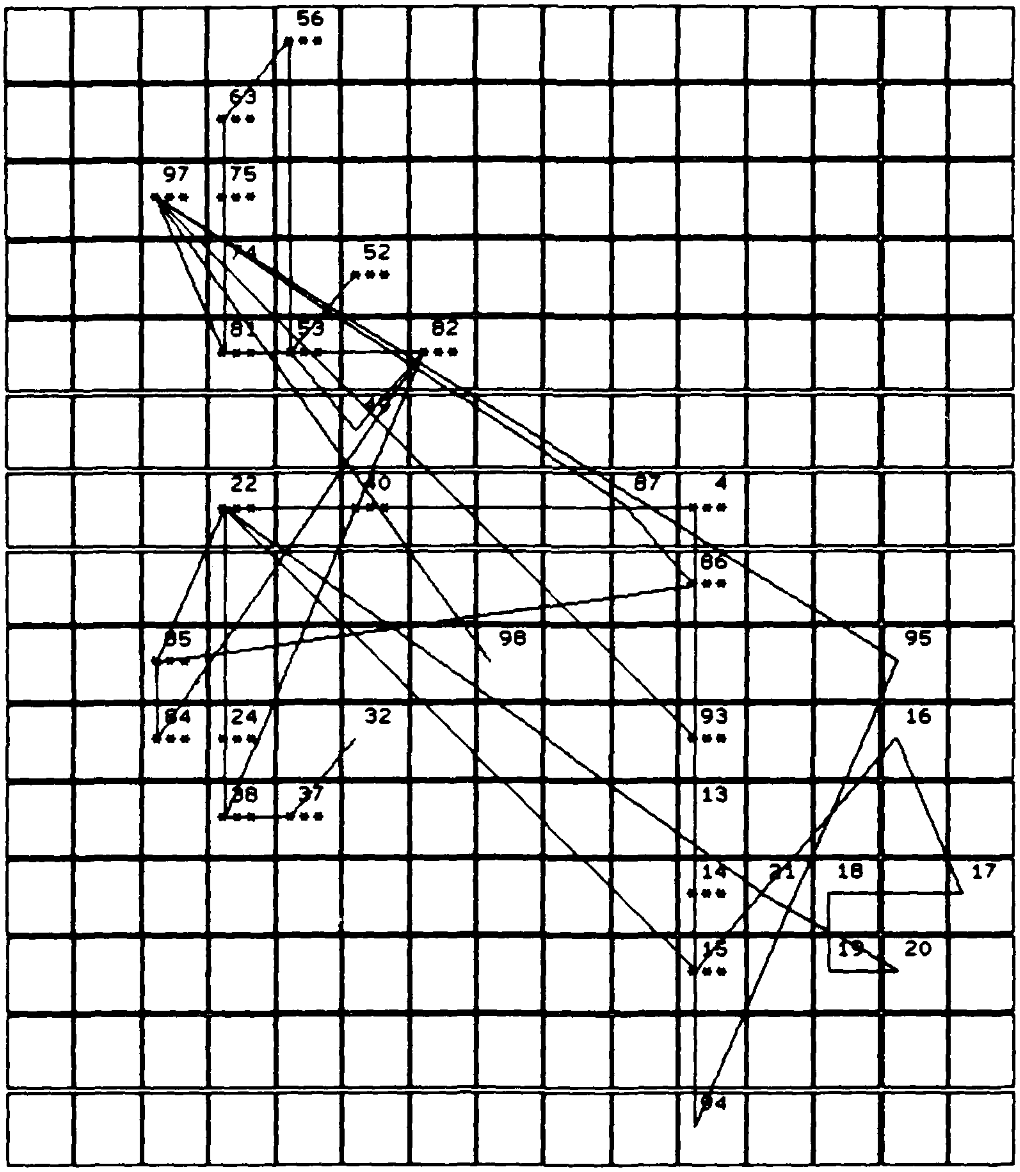


Figure 4.2. Net Speak10 Trained with Conscience = 1.5. Without prior knowledge, there is no way to visually determine the amount of conscience used to train a net.



zero5.trn --> speak10.net

Figure 4.3. Full Trajectory of the Word Zero. The numbers represent the last time slice to light a node. When a node is lit more than once, three asterisks are printed.

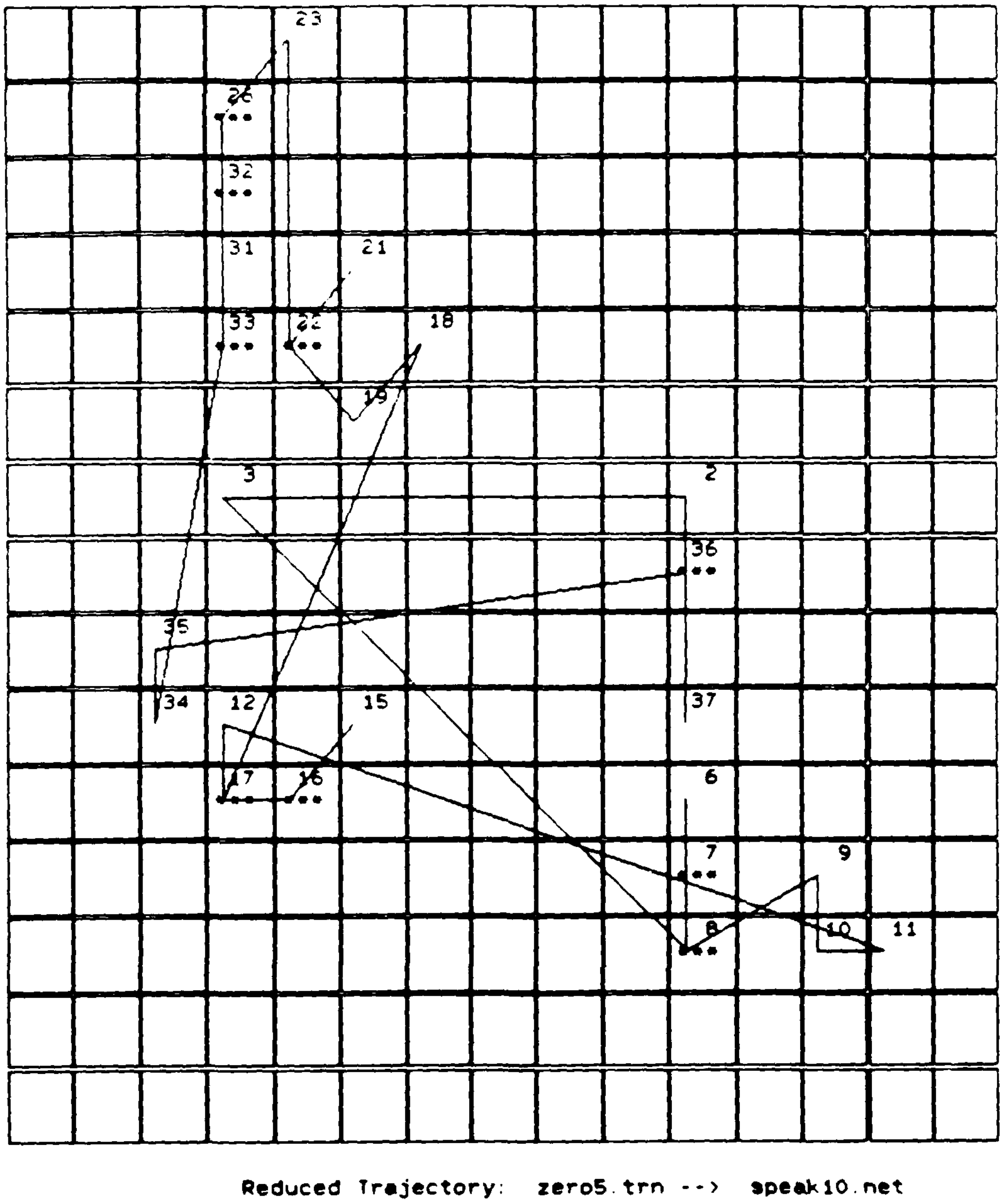


Figure 4.4. Reduced Trajectory of the Word Zero. Transitory points not near other points are eliminated. The "slice" numbers shown are actually the respective location of a lit node in the reduced trajectory. Note the much simpler curve when the trajectory is reduced.

```
zero5.trn -->  speak10.net

Trajectory through map:  (98)


145 115 100 100  93 122  93  93  93 190 190 175 160 175 190
148 179 177 192 193 176  93 138 138 153 153 154 154 154 154
154 140 154 154 154 154 154 153  95  95  66  66  66  66  66
 66  66  66  80  64  50  50  64   4   4   4  18  33  33  33
 33  33  18  33  33  33  33  33  33  63  33  33  63  48  33
 63  63  32  63  63  63  66 137 137 122 115  99  32 145 145
145 145 145 220 133  32  32 127

After elimination of transients (89) :


145 115 100 100  93  93  93  93 190 190 175 160 175 190 179
177 192 193 176 138 138 153 153 154 154 154 154 154 140 154
154 154 154 154 153  95  95  66  66  66  66  66  66  66  66
 80  64  50  50  64   4   4   4  18  33  33  33  33  33  18
 33  33  33  33  33  33  63  33  33  63  48  33  63  63  63
 63  63 137 137 122 115  99 145 145 145 145 145  32  32

(Reduced Trajectory) Only three in a row! (81)

115 100 100  93  93  93  93 190 190 175 160 175 190 177 192
193 138 138 153 153 154 154 154 154 154 140 154 154 154 154
154 153  66  66  66  66  66  66  66  66  80  64  50  50  64
  4   4   4  18  33  33  33  33  33  18  33  33  33  33  33
 33  63  33  33  63  48  33  63  63  63  63  63 137 137 122
115 145 145 145 145 145

Now listing final trajectory! (37)

115 100  93 190 175 160 175 190 177 192 193 138 153 154 140
154 153  66  80  64  50  64   4  18  33  18  33  63  33  63
 48  33  63 137 122 115 145
```

Figure 4.5. Reduction Process for the Word Zero. Each point in a trajectory represents a node. "0" is the upper left node, "14" is the upper right, and "224" is the bottom right node.

Table 4.1. Presentation of Inputs

Net	Process	Isolated Words	Connected Words
speak1	sequential	95%	75%
speak3	random	90%	73%
speak4	random word	100%	56%

Note: Because sequential training resulted in overall best performance, it is used to train all later nets.

Dynamic Time Warping

Dynamic time warping tests were performed over a long period of time. The earliest tests used a smaller set of test words and in general evaluated different aspects of the system. Each evaluation is described below.

Training Process. One of the first uses of DTW was to evaluate the first Kohonen net's training. In this instance, the DTW testing was not yet automated. Ten arbitrary digits were used for templates. Ten different digits were used to test for isolated speech recognition, and eight utterances of connected digits (containing another 48 digits) were used to test for continuous speech recognition. The connected digits at this point were intentionally slurred together as much as possible. The DTW stretch factors were set at 1.0, 0.5, and 0.5.

The three types of training processes evaluated in Table 4.1 vary the presentation of the inputs to the net. The first method takes an utterance containing all ten digits, with pauses between them, and repeatedly presents the input vectors to the net in their natural sequence (called sequential training). The second method presents the input vectors from the same utterance to the net in a random sequence (called random training). The last method takes 10 utterances of individual digits and presents them to the net in a random sequence. Individual vectors within the randomly selected word are presented in their natural sequence (called random word training).

No conscience is used in any of these nets. The sequential training process appears to provide the overall best performance. The reasons behind the relative performances were not investigated. However, it was noted that the random number generator did not produce a very uniform dis-

Table 4.2. Comparison of Stretch Factors

Stretch Factors	Isolated Words	Connected Words
2.0, 0.5	75%	80%
1.5, 0.5	75%	80%
1.0, 1.0	100%	73%
1.0, 0.75	90%	80%
1.0, 0.5	80%	84%
0.75, 0.75	100%	79%
0.75, 0.5	90%	76%
0.5, 0.5	100%	71%

Note: The 100% isolated rate for equal stretch factors confirms that isolated words were spoken at the same speed as the templates. The connected utterances were actually spoken faster.

tribution. Since sequential training worked best, whatever the reason, it is used in all later net training.

Stretch Factors. Variations in performance, as the stretch factors were changed, was observed during development and debugging of the DTW routines. This suggests that additional testing, where only the stretch factors are changed, could provide the best performance for the given set of utterances.

For the tests shown in Table 4.2, the best performing net at that time, "speak1", was used. Only, the stretch factors in the right and up directions, respectively, are varied. The diagonal stretch factor remains 1.0.

Note the tradeoff between results for isolated and connected speech. The fact that 100% accuracy is obtained for isolated speech whenever the off-diagonal stretch factors are equal implies that the isolated words under test were spoken at the same speed as the templates. Likewise, when the stretch factors are not equal, recognition of the faster spoken connected utterances improves.

At this point, comparisons of the envelopes of the connected speech utterances used here with those used by Dawson (13) were made. It was found that Dawson clearly separated his words in the continuous speech utterances. For ease of comparison, a new set of connected utterances

Table 4.3. Tests of Clearly Spoken Connected Speech

Stretch Factors	Connected Words
1.0, 0.5	90%
0.75, 0.75	91%

Note: Connected speech recognition improves when the words are spoken clearly and are not slurred together.

Table 4.4. Conscience in First Kohonen Nets

Net	Conscience	Isolated Words	Connected Words
speak1	none	90.0%	93.0%
speak9	1.1	82.7%	72.1%
speak10	1.5	99.1%	90.7%

Note: A modest application of conscience (1.5) results in a large improvement in isolated recognition, while too much conscience (1.1) degrades performance.

were generated in which the words were not slurred together. The results from these tests, using "speak1", are shown in Table 4.3.

The improved results are dramatic and are now comparable to the results Dawson obtained. Therefore, this set of connected utterances, as well as the stretch factors of [1.0,] 0.75 and 0.75, are used in all later tests.

Conscience. After finding that conscience assisted in training the second Kohonen net (see the following sections), new first Kohonen nets were trained. Table 4.4 shows the results of DTW runs on those nets. For these (and later) tests, the number of isolated words being tested was increased to 110.

Note that while conscience dramatically helps recognition of isolated words (when there is not too much of it), it also slightly degrades recognition of connected words. During the training run on "speak9", it was found that only 25 to 40 percent of the nodes were eligible each training cycle (because of conscience) for matching the given input. Given the accuracies in Table 4.4, one could infer that nodes are being under or improperly trained with a conscience factor of 1.1.

Templates. Each of the prior DTW tests used 11 arbitrary templates—one for each of the ten digits and one for silence. With the variability in trajectories between different instances of the same digit, selecting specific templates might improve the recognition rate.

Thus, unsuccessful attempts were made to increase accuracy by using various template selecting algorithms. One algorithm, that seems quite reasonable, resulted in the greatest degradation. Twenty examples of each of the ten digits were collected. The one example out of the twenty that had the lowest average DTW minimum path to the other nineteen was selected as the template for that digit. Using these newly selected templates, stretch factors of 0.75 and 0.75, and net "speak10", isolated recognition dropped from 99.1% to 94.5%. The connected digit recognition rate dropped more—from 90.7% to 76.7%.

A small amount of degradation might occur because of a change in relative stretches between the templates and the utterances under test. However, the connected rate drop of 14% seems excessive. Apparently, the tested digits were not closest to the "average" template from each set of 20.

Speaker Independent Speech Recognition. A small number of tests were performed on utterances spoken by someone other than the net trainer (the speaker whose utterances trained the Kohonen net). A deep female voice was used for these tests; whereas all other tests used a mid-range male voice.

A set of 10 isolated digits and eight connected digit utterances (containing another 48 digits) were tested using the templates in the male voice, net "speak1", and stretch factors of 0.75 and 0.75. The results were a 40% isolated and 31.3% connected digit recognition rates. It is expected that if templates were generated from the female's speech, the results would improve.

Second Kohonen Neural Net

Extensive tests were run on a number of forms of the second Kohonen neural net. The driving motivation was the possibility of decreasing computation time and providing a system where all major components emulated simplified neural activity

Early Attempts. Limited development took place for two types of inputs described in Chapter III. In both of these, an input vector had 225 components. In both cases each component represented one of the 225 nodes in the first Kohonen net. In the first case, a component was incremented (from zero) any time its respective node was lit in a word's trajectory. In the second case, the "activation level" of each node was added to that node's cumulative activation from every input (slice) in an utterance. Given a vector input to the first net, the activation level for a node is the dot product of that node's weight vector with the input vector.

The possible advantage of these types of input is that they can be normalized, allowing simple and fast computations in using and training a net. However, it deletes the time dependent relationship of the various sounds in a digit.

Ninety words (nine versions of each digit) were used to train these nets. After training, the 90 words were input to the net and the nodes that lit up were identified. Unfortunately, less than half a dozen nodes lit up in each case. At that point, these nets were considered unsuccessful, and development of other forms proceeded.

Yet, there are at least two reasons why the labeling of these nets as *unsuccessful* is questionable. First, it was later noted that training and testing of the second Kohonen net took place using dot product algorithms rather than Euclidean distance. Since the inputs at this point were ***not*** normalized, dot products could not give valid results. Secondly, conscience had not yet been implemented. It is quite possible that assigning random numbers to the initial weights gives only a few nodes with vectorial representations even close to the cumulative activation surfaces described above. Both reasons suggest that some additional testing might prove beneficial.

Table 4.5. Conscience in Second Kohonen Nets

Net	Conscience	Nodes Lit	Differentiation
path5	none	58%	91.1%
path11	1.5	81%	98.9%
path12	1.1	89%	98.9%

Note: These results show that conscience increases the effective number of nodes that "look" like the training inputs. They do not necessarily imply comparable performance in recognizing words.

Trajectory Input Nets. As described in the prior chapter, two alternate forms of input were also explored. These consist of trajectories represented either as 100 scalar values or 100 x-y coordinate pairs. Shorter trajectories are filled with trailing −1's. However, initial development, as described below, used a 75 scalar input (trajectory of length 75 or less) with trailing zeros. It will later be seen that no significant improvement or degradation in accuracy arises from switching between 75 and 100 point trajectories.

Conscience. As mentioned, initially a 75 point trajectory was selected. Unfortunately, after training (without conscience), only about 58% of the nodes lit up when the set of training digits were input. In addition, when looking at the nodes that lit up, only 91% of the 90 training digits could be correctly differentiated.

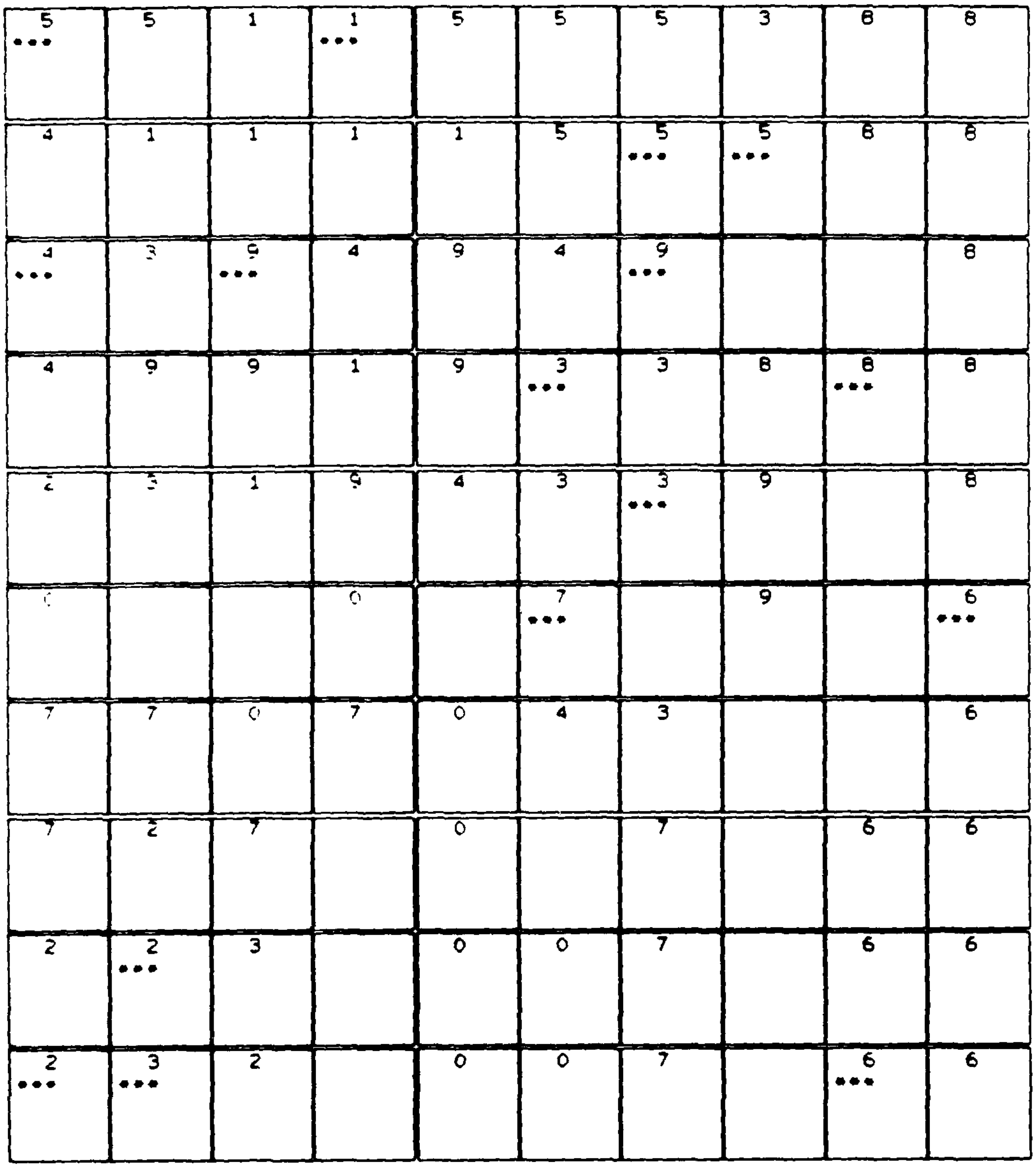
Figures 4.6 through 4.8 show the result of nets trained with: (a) no conscience, (b) a conscience factor of 1.5, and (c) a conscience factor of 1.1. Each of these nets used 75 scalar inputs. Note that the numbers represent which training digit lit up the respective node (only 90 training digits were used for these tests).

Table 4.5, derived from the tests which generated these figures, shows numerically the advantages of using conscience. Of course, these tests still do not show Kohonen net performance on a different set of digits than those used in training.

9 •••	9		4 •••	4	5 •••		1 •••		9 •••
		4 •••		3		1	9 •••	1	
7 •••	7	2		9			1		9 •••
0			3	4	9 •••	9	1 •••		4
0		0	7					5	5
0	0	7		7 •••	3	3 •••	5 •••		5
		7		2					5 •••
6 •••			2	3 •••	3 •••	2 •••	8	3	
6		2 •••		2					8 •••
6 •••		7 •••		7		8 •••		8 •••	8

twopic.hdr -> speak1.net -> path5.net

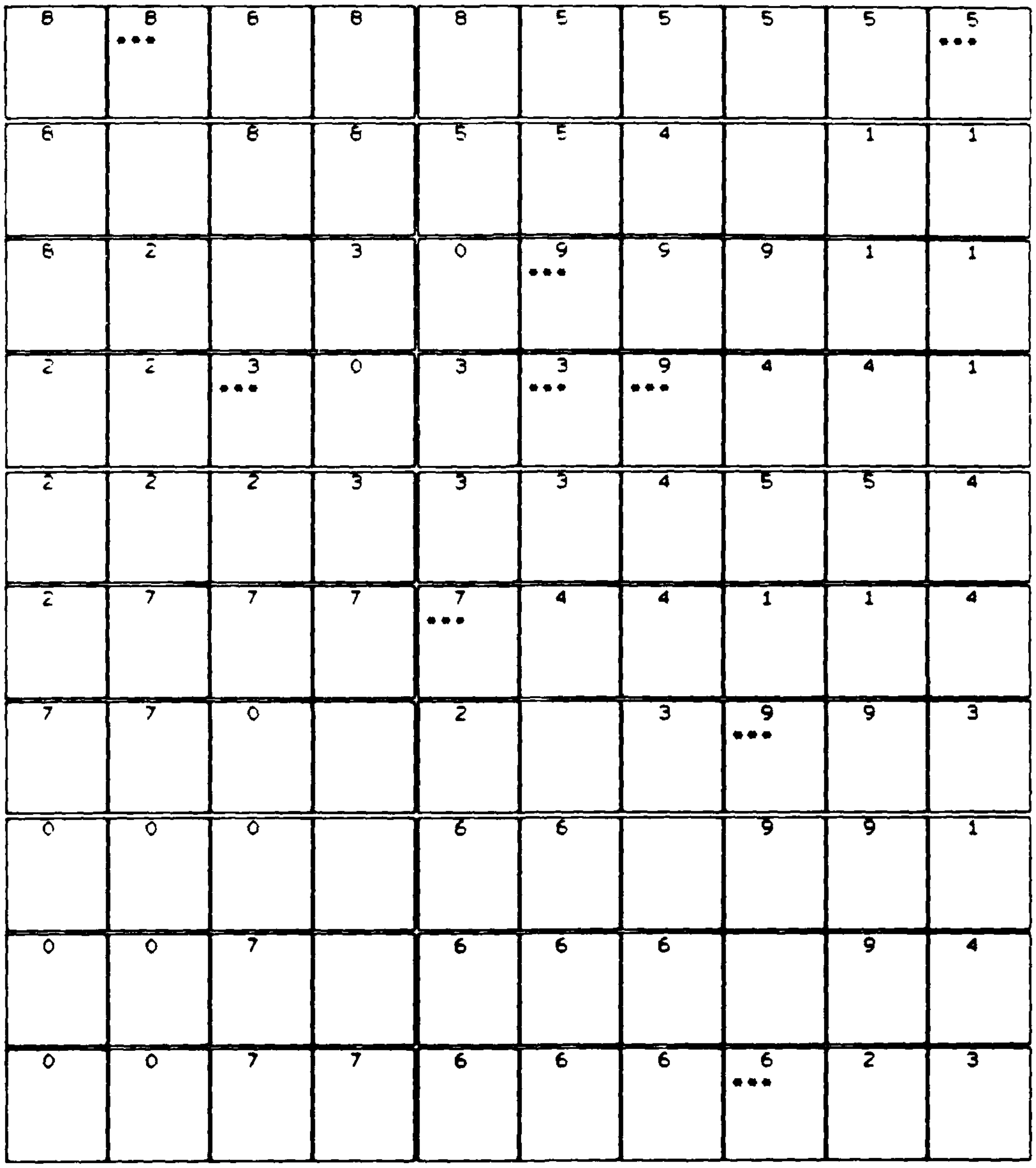
Figure 4.6. Net Path5 Trained with No Conscience. Note that only 58% of the nodes are lit up when the *training digits* are applied at the inputs.



5 •••	5	1	1 •••	5	5	5	3	8	8
4	1	1	1	1	5	5 •••	5 •••	8	8
4 •••	3	9 •••	4	9	4	9 •••			8
4	9	9	1	9	3 •••	3	8	8 •••	8
2	3	1	9	4	3	3 •••	9		8
0			0		7 •••		9		6 •••
7	7	0	7	0	4	3			6
7	2	7		0		7		6	6
2	2 •••	3		0	0	7		6	6
2 •••	3 •••	2		0	0	7		6 •••	6

```
twopic.hdr -> speak1.net -> path11.net
```

Figure 4.7. Net Path11 Trained with Conscience = 1.5. With a moderate amount of conscience, 81% of the nodes light up when the *training digits* are applied at the inputs.



8	8 •••	6	8	8	5	5	5	5	5 •••
6		6	6	5	5	4		1	1
6	2		3	0	9 •••	9	9	1	1
2	2	3 •••	0	3	3 •••	9 •••	4	4	1
2	2	2	3	3	3	4	5	5	4
2	7	7	7	7 •••	4	4	1	1	4
7	7	0		2		3	9 •••	9	3
0	0	0		6	6		9	9	1
0	0	7		6	6	6		9	4
0	0	7	7	6	6	6	6 •••	2	3

```
twopic.hdr -> speak1.net -> path12.net
```

Figure 4.8. Net Path12 Trained with Conscience = 1.1. When a large amount of conscience is used, 89 nodes light up when the *training digits* are applied. This does not necessarily imply a commensurate improvement in word recognition performance.

Table 4.6. Scalar Input Net Tests

Method	Accuracy
Euclidean search of training set	42%
DTW search of training set	90%
DTW search of non-training set	91%

Note: This implies that "recognizing" a node's weight vector as a given digit is ineffective when the stretch variable is not taken into account.

Performance of 75 Scalar Input Nets. Initially, it was assumed that identification of a node with the training digit that caused it to light up was the proper way to use a net. That is, input an unknown (non-training) digit, see which node lights up, and identify the unknown digit as the training digit associated with that node.

Euclidean distance is the algorithm first used to identify a lit node. But this achieved only a 42% correct identification of a set of 100 test (non-training) digits. Possibly, this resulted from the non-uniform stretch inherent in trajectories. Accordingly, a DTW algorithm was substituted for Euclidean distance (in test only, not training). This resulted in a 90% correct identification of the test digits.

However, it is possible for a node's weight vector to be "closer" to a training digit (say digit A) which did not cause the node to light up than the training digit that did (e.g., digit B). This occurs when two nodes should be lit up by digit A, but the software allows only one node to light up (let it be the "first"). The problem occurs when the activation level of the "second" node, from digit B, is less than its activation level from digit A. Because of the software, the second node is lit by B, when in actuality its weight vector is closest to the input vector from digit A. This problem was corrected by searching through a set of 100 non-training digits to find the closest one to each node. This increased identification to 91%. Table 4.6 summarizes the above findings. All entries use the net "path12" trained with a conscience factor of 1.1.

With this data, one can now compare DTW with a second Kohonen net as word recognition

Table 4.7. Conscience in 100 Scalar Input Nets

Net	Conscience	Accuracy
path16(v1)	1.1	86%
path16	1.5	92%

Note: Too much conscience (1.1) again reduces performance.

algorithms. With the DTW routine achieving up to 99% accuracy, the 91% seen here is not impressive.

Performance of 100 Scalar Input Nets. After noting that a substantial number of the digits had reduced trajectories between 75 and 100 points, the routines were updated for 100 scalar inputs. Additional tuning also took place in the DTW routine that replaced Euclidean distance. The minimum path distance was normalized (divided by the number of elements in the path) to give a local average path length. This replaced simply dividing by the number of node weights not (trailing) zeros. Another change was to use the length of both the trajectory and the number of node weights that were not trailing zeros. Prior to this point, the number of a node's weights not zero was used as the DTW length for both the node's weight vector and the word under test.

Table 4.7 shows the results of the longer inputs and retuned DTW as the amount of conscience in the second Kohonen net is varied. Unfortunately, the changes resulted in an accuracy only 1% better than the best 75 scalar input net. However, it shows again that too much conscience degrades performance. These tests used "speak10" as the first Kohonen net.

Performance of 100 X-Y Pair Input Nets. Since the scalar input nets provided only 92% accuracy, and using mini-DTW rather than a Euclidean distance increased the computation time, another approach was tried. If the trajectories remained a sequence of x–y coordinate pairs (instead of being translated to a sequence of scalar values), mini-DTW might not be necessary. Also, the training effectiveness might improve if the 15 unit difference in the scalar representation

Table 4.8. Accuracy of 100 X-Y Pair Input Nets

Net	Conscience	DTW	TAXI
path21	1.1	86%	84%
path22	1.5	86%	85%
path23	none	73%	69%

Note: These results can not be compared to that of scalar nets since the set of training digits was changed. Also, the improvement from DTW over TAXI distance is very slight in these tests.

Table 4.9. Differently Trained 100 X-Y Pair Input Nets

1st Net	2nd Net	Conscience	DTW	TAXI
speak10	path26	1.5	85%	80%
speak1	path27	1.5	91%	90%

Note: When the training set is the same for both scalar and x-y pair nets, the performance is comparable—91% to 92% .

of vertically adjacent nodes is absent.

Thus, training and test routines were written to use 100 x-y coordinate pair inputs and both DTW and Euclidean (actually TAXI) distance algorithms. Table 4.8 compares the performance of nets using both algorithms as conscience is varied. During these tests, the training and test sets of digits were changed. Both sets used only 90 digits and one word of silence. Also, all of the second nets were trained using "speak10" (conscience at 1.5) for the first net.

These tests suggest that using mini-DTW (in x-y pair nets) to identify a word provides only marginal improvement over using TAXI distance. However, the results here do not justify going from scalar to x-y pair inputs.

Remember though, the training and test sets were changed slightly. Thus, to prevent questions about the validity of comparing these accuracies to those from the scalar nets, two other nets were trained. These used the same sets of 100 training and test digits as "path16" (the best scalar input net). Table 4.9 shows the results; the conscience factors listed are for the second net.

Table 4.10. Scalar Input Nets Without Training

First Net	No. Inputs	Conscience	DTW	TAXI
speak1	75	none	96%	88%
speak10	100	1.5	93%	86%

Note: Training, with the procedure and training data set used, was actually detrimental to net performance.

An interesting point here is that "speak1", trained without conscience, resulted in the best performance. Also, it should be pointed out that the 91% achieved here is *not* significantly different than the 92% obtained for scalar input nets.

One should note that in the test of "path26", 8 of the 15 errors had zeros labeled as twos on node number 16. Yet no twos were closest to that node. The logic is somewhat complex, but this suggests the possibility of improvement to 93%. Unfortunately, it also emphasizes the nonuniformity of trajectories within classes (of digits) and the need for supervised learning in the first Kohonen net.

Second Kohonen Nets Without Training. As was mentioned earlier, the second Kohonen net is essentially a codebook of trajectories (as is, really, any Kohonen net). So, what would be the result if the training set of 100 digits was simply assigned to the 100 nodes and no training performed? At the very least, one would obtain a baseline from which the effectiveness of training could be assessed.

This approach is feasible for the second Kohonen net since the number of training inputs is the same as the net size. For the first Kohonen, there were too many inputs.

Table 4.10 shows the results of assigning the zeros to the first row, the ones to the second, etc. The results shown use scalar inputs and vary the training used in the first neural net. Both the training (now just assignment) and the test sets of digits contained 100 words without silence. The conscience listed is that used in the first Kohonen net.

This shows that the training employed here does not improve recognition accuracy of scalar

Table 4.11. X–Y Pair Input Nets Without Training

1st Net	No. Inputs	Conscience	DTW	TAXI
speak1	75	none	95%	88%
speak1	100	none	95%	88%
speak10	75	1.5	93%	85%
speak10	100	1.5	93%	85%

Note: Again, there is no significant difference between scalar and x-y pair nets. Neither is there any difference when the trajectory length is 75 or 100 points.

input nets. But what does training do for x–y pair input nets? Table 4.11 shows the results of tests where the inputs are x–y pair reduced trajectories. The data show the significance of the trajectory length (75 or 100). Again, the conscience listed is for the first Kohonen net.

Clearly, the trajectory lengths used here do not affect the results. Also, it appears that with the current forms of training and training data sets, training does not result in optimum performance for either scalar or x–y pair input nets!

Connected Speech Recognition. A limited amount of development and testing was performed using the second Kohonen for recognizing continuous speech. The initial results were unsatisfactory and computationally intensive. Accordingly, effort was discontinued and the primary emphasis was placed on isolated digit recognition.

However, the results do imply some promise if the productivity of the first and second nets is improved. The basic routine takes a 100 slice (partial trajectory) window from a continuous utterance and compares it, using the DTW algorithm, with all the nodes' weight vectors in a second Kohonen net. As described earlier, a digit is selected for that window. The window is also assigned a distance which is the normalized minimum path length—the smaller the better. The window is moved one slice at a time from the start of the utterance to the end. This results in a sequence of digits and path lengths where the sequence length is the number of slices in the utterance. When the path length becomes a minimum, the beginning of a new digit (the one assigned to that slice) occurs.

The second Kohonen used for the test was an untrained one developed from the trajectories through the first net, "speak1". This had the highest isolated digit recognition rate at that time. Two utterances containing a total of 16 digits were tested. One utterance was slurred together; the other was spoken distinctly. The recognition rate was 81.25%.

Several simple, common sense rules were used to eliminate brief local minima. Additionally, the computation time can be significantly reduced by anticipating the minimum number of slices between words and using a gradient search algorithm rather than computing the comparison for every slice. Because of the unsatisfactory recognition rate, the gradient search routine was not implemented.

Summary

This chapter has quantified the performance of the speech recognition system developed as well as the factors involved in its performance. Tradeoffs show the system works best with sequential training of the first net, a conscience factor of 1.5, and DTW as the word recognition algorithm. DTW in turn works best (for the utterances tested) with stretch factors of 1.0, 0.75, and 0.75.

When a second Kohonen net is substituted for DTW, performance falls. It doesn't seem to matter whether scalar or x-y pair trajectories are used, or the length of the trajectories. Unfortunately, training the second net, rather than simply assigning arbitrary trajectories to the nodes' weight vectors, seems to further degrade the system.

These results, and their implications, are summarized in the next chapter.

V. Conclusions and Recommendations

A large number of programs were developed and tested to demonstrate the feasibility of using neural nets and dynamic time warping in speech recognition. Some concepts proved to be completely unworkable and are not mentioned here. Others showed some promise, but were neglected for development of more promising approaches.

The most successful algorithms were the generation of net trajectories and one pass dynamic time warping. But because of the intense computations required by DTW, much effort was spent in developing a second Kohonen net to replace it. Unfortunately, the second Kohonen net implemented here does not perform quite as well as DTW.

Conclusions

The preprocessing, first net training, and trajectory reduction algorithms apparently are adequate to assure good isolated and connected digit recognition rates in speaker dependent speech. Using *conscience*, rates of 99.1% and 90.7%, respectively, can be achieved from a one pass dynamic time warping algorithm. A second neural net can achieve up to a 96% isolated digit recognition rate (but only 81% for connected speech).

Dynamic Time Warping. The computation and time requirements were reduced when the DTW input vectors (points on trajectories) were two dimensional rather than the more traditional 16 to 256 dimensional (possibilities from FFT routines).

However, there are some deficiencies in the system. First, the DTW accuracy appears strongly dependent on the choice of templates. Secondly, trajectories within a digit's class are not uniform enough to provide excellent results. This appears to be a problem inherent in Kohonen nets trained without supervision. They simply do not group *all* of the members of a class in the same region.

The latter problem appears to have two causes. The most obvious is the situation where a class (a particular sound) consists of members in two disjoint regions in 15 dimensional hyperspace. The other cause is the way that initial weights are assigned and then updated. A small, but significant percentage of nodes, without supervision, can simply not be correctly trained. Correction of either of these problems should improve consistency and performance.

However, any DTW algorithm will always be computationally intensive. As such, it is extremely limited when a larger vocabulary is needed.

Second Kohonen Net. Using a second Kohonen net in place of DTW appears to have possibilities. Such a system can achieve at least a 92% isolated speech recognition rate (using trained nets) or over 96% when the net acts as a codebook (no training).

The fact that untrained nets produced better results implies that either training produces weights not representative of the data set to be recognized (i.e. the training equations are somehow defective) or that the training procedure is in error. The latter is the most likely conclusion; most probably, there was simply *not enough data* to effectively train this size net.

Irrespective of the reason for getting better results with untrained nets, the accuracy of the trained nets is satisfactory to evaluate their performance. Thus, the following thoughts address, for the most part, trained nets.

The accuracy of the net does not depend significantly on the form or length of the word trajectories. Routines were developed where the trajectories were represented both as sequences of scalar values and as sequences of x-y coordinate pairs (the two possible forms). Although the scalar representation was expected to have more difficulties (because of the 15 unit distance between vertically adjacent nodes), the achieved accuracy was actually 1% better[10] than the x-y pair representation. The scalar versions achieved 92% accuracy as compared with 91% for the x-y pair.

However, there are drawbacks to using a second Kohonen net with trajectory inputs even if the accuracy is improved. The non-uniform stretch in words requires using a mini-DTW algorithm

[10] 1% can not be considered a significant difference.

rather than a Euclidean distance (to compare and identify nodes and utterances) to achieve the best performance. Using a mini-DTW, such a net could not be efficiently implemented in hardware. When Euclidean (actually TAXI) distance is used, the recognition rate drops by on. to several percentage points. Further, a dot product is the preferred algorithm over Euclidean or TAXI distance, but dot products require normalized arguments. The scalar and x-y pair trajectories used as inputs here can not be normalized without destroying their information content. Alternate forms of inputs, that could be normalized, were not pursued long enough to determine their feasibility.

Recommendations

There are several quite obvious tests that should be run. These include speaker independent tests, larger vocabularies, and alternate (or multiple) feature sets. However, there are a few basic changes to look at first. The most important of these is to obtain consistent in class trajectories.[11] Supervised learning of the first Kohonen could limit the locations of sounds, within a sound class, to a particular region. Kohonen's learning vector quantization (LVQ) algorithm (19) could perform this task.

After supervised learning optimizes the trajectories, the preprocessing procedure might be retuned to eliminate any unnecessary processing. This could be done by repeatedly making a change and observing the results on the automatic DTW tests.

To pe.mit larger vocabularies, and possibly speaker independence, the use of multiple feature sets might be investigated. Dawson (13) and Kim (20) were successful at using a combination of linear predictive coefficient (LPC) spectra, zero crossing rate, and frication frequency to identify speaker independent speech. This might require two additional very small nets (possibly only one-dimensional), with a small increase in processing time, to provide a large jump in performance.

If necessary, using multiple nets or templates might allow recognition of speaker independent

[11] "In class" here refers to the set of trajectories for any given word. The current net produces out of class trajectories that are separable, but does not produce in class trajectories that "look" alike.

speech. Tests should be run on multiple sets of utterances from various speakers and a larger vocabulary. Kim (20) collected such a set of data.

Finally, in place of the second Kohonen nets developed here, one might try other algorithms. A backward propagation net might work very well. Alternatively, one might delve further into the use of activation surfaces as inputs to a second Kohonen net rather than simple trajectories. If a second net becomes effective, it should be examined for use in identifying connected speech.

Summary

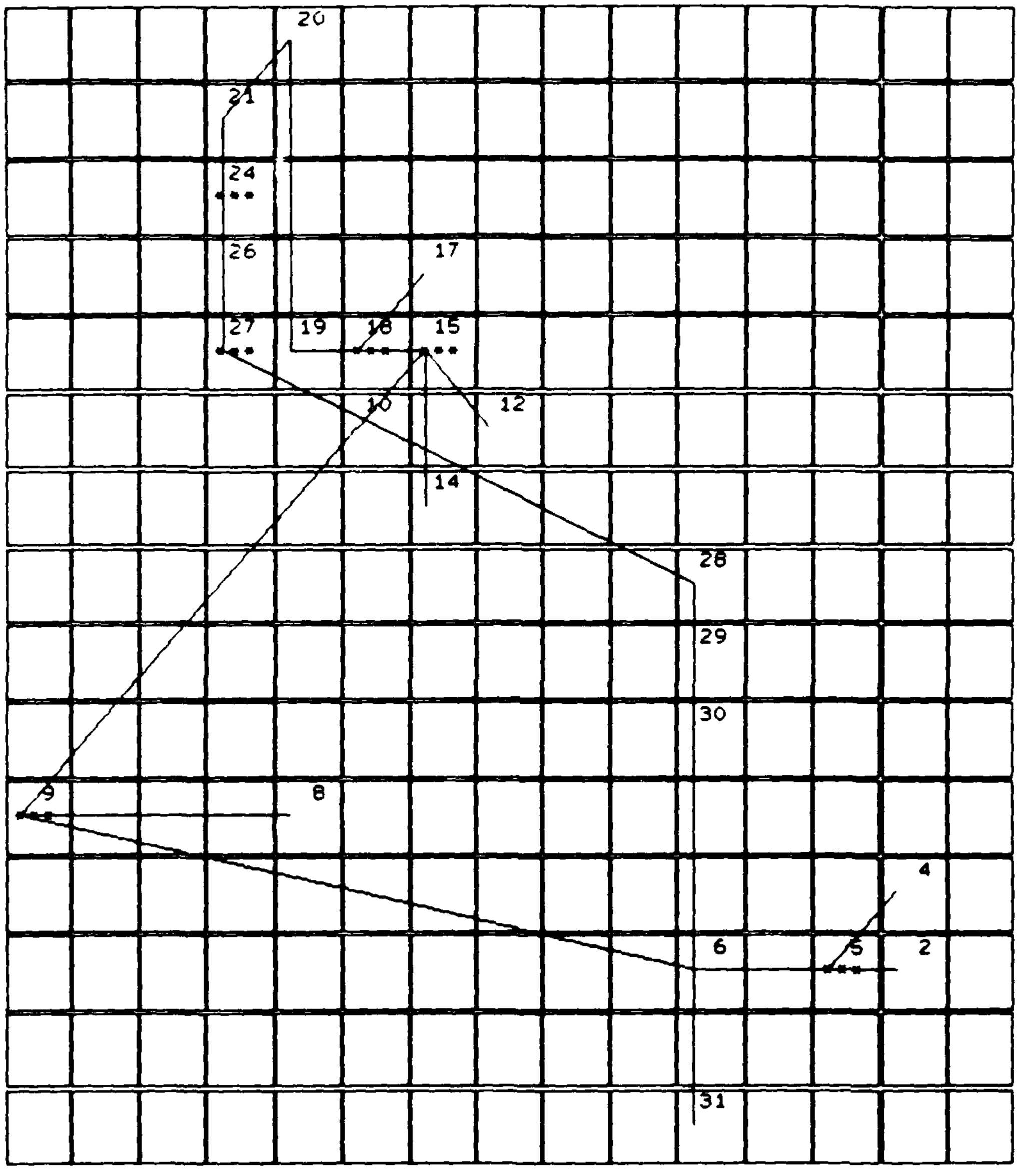
The purpose of this effort was to show the feasibility of using neural nets in speech recognition. For speaker independent utterances from a small vocabulary, that was done.

Isolated digit recognition was achieved at up to 99.1% and connected speech recognition at 93%. While this used only a small vocabulary, it also used relatively simple feature sets and templates chosen without significant tuning.

By adding such characteristics as supervised learning and multiple feature sets or nets, the prospect is bright that this approach could successfully handle larger vocabularies and speaker independence while keeping additional computation time to a minimum.

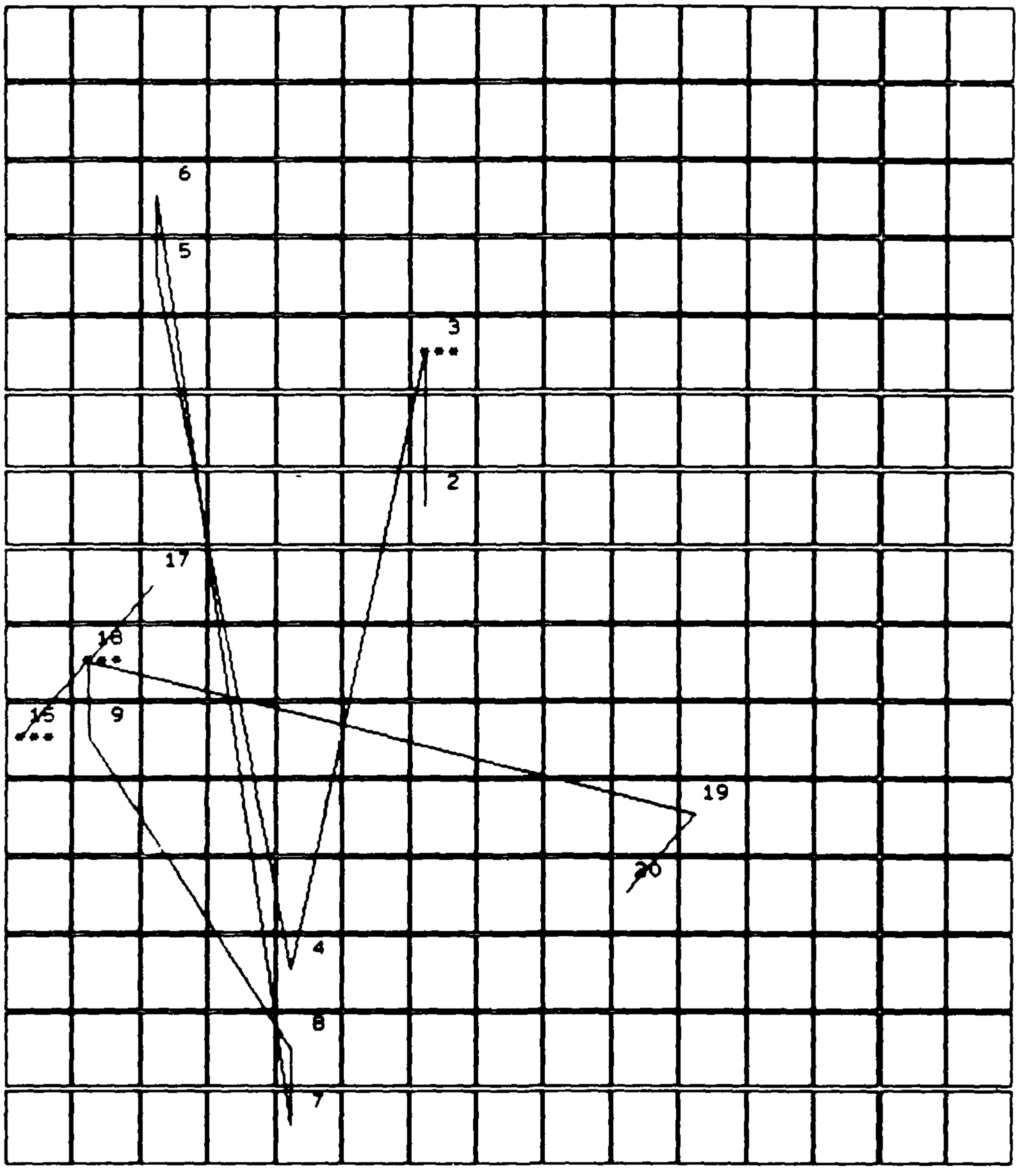
Appendix A. Template Trajectories

The following figures show the reduced trajectories for the eleven templates used in the DTW tests. The templates include the digits zero through nine and a short period of silence.



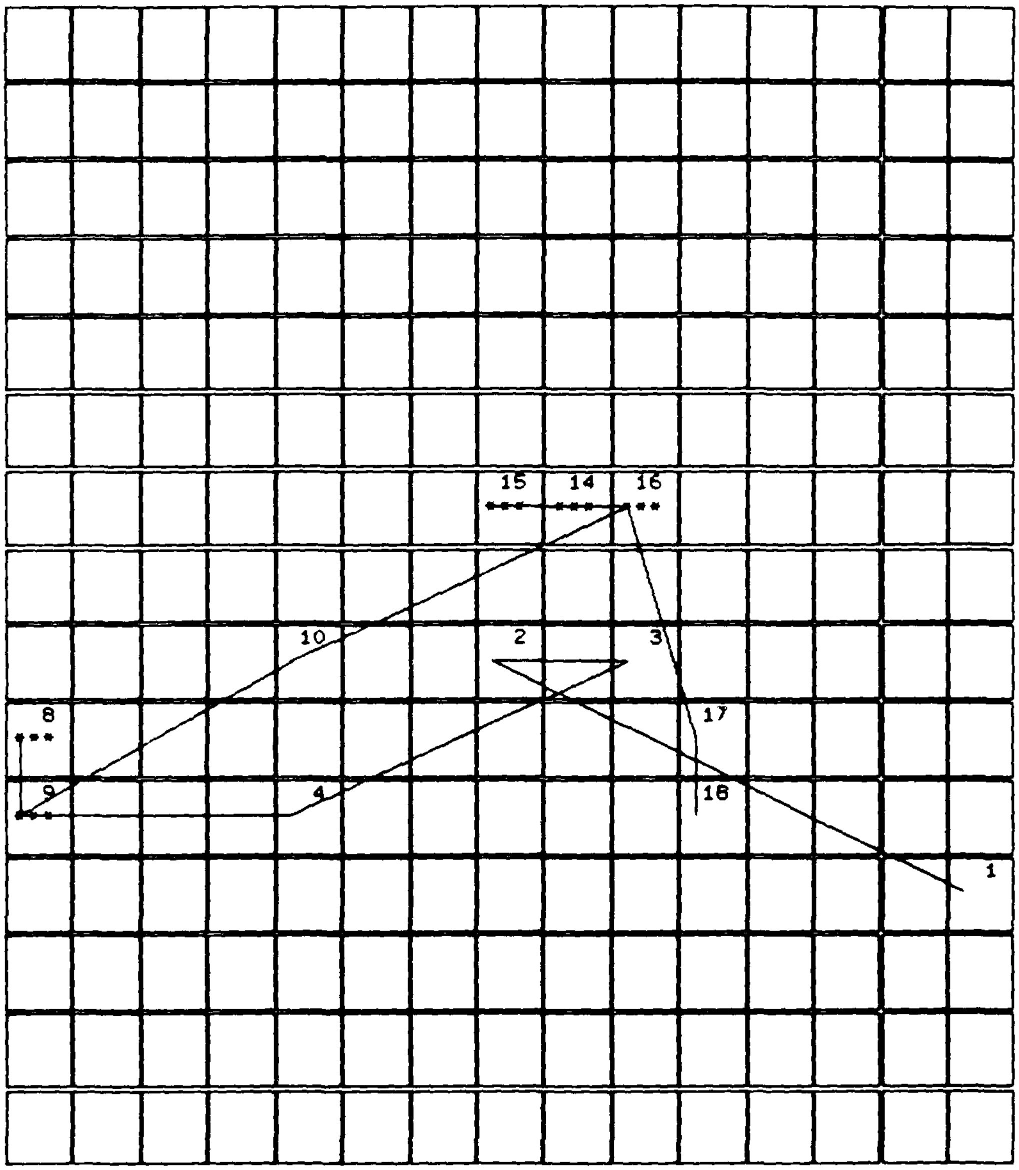
Reduced Trajectory: zero0.trn --> speak10.net

Figure A.1. Reduced Trajectory of the Template Zero



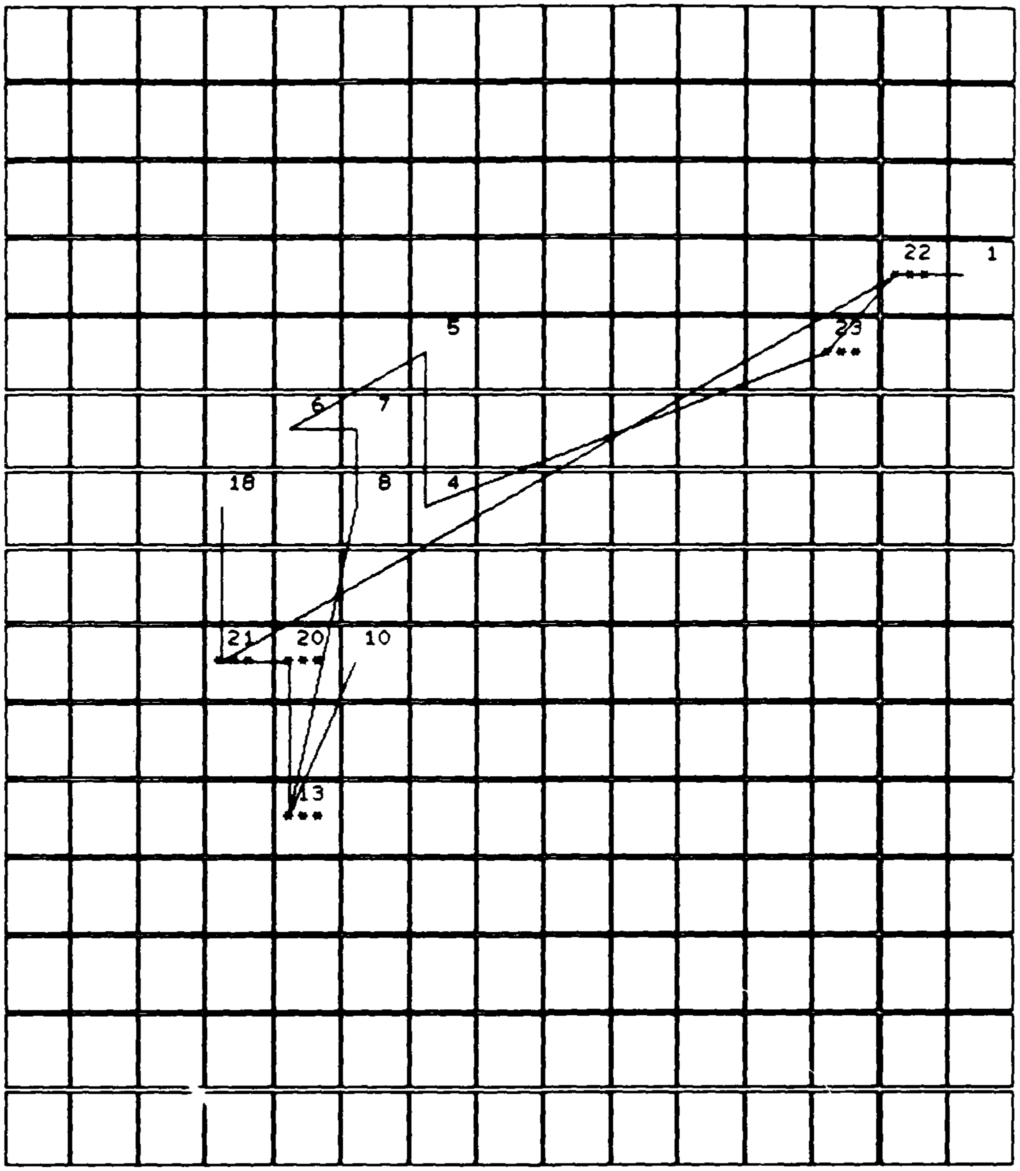
Reduced Trajectory: one0.trn --> speak10.net

Figure A.2. Reduced Trajectory of the Template One



Reduced Trajectory: two0.trn --> speak10.net

Figure A.3. Reduced Trajectory of the Template Two



Reduced Trajectory: three0.trn --> speak10.net

Figure A.4. Reduced Trajectory of the Template Three

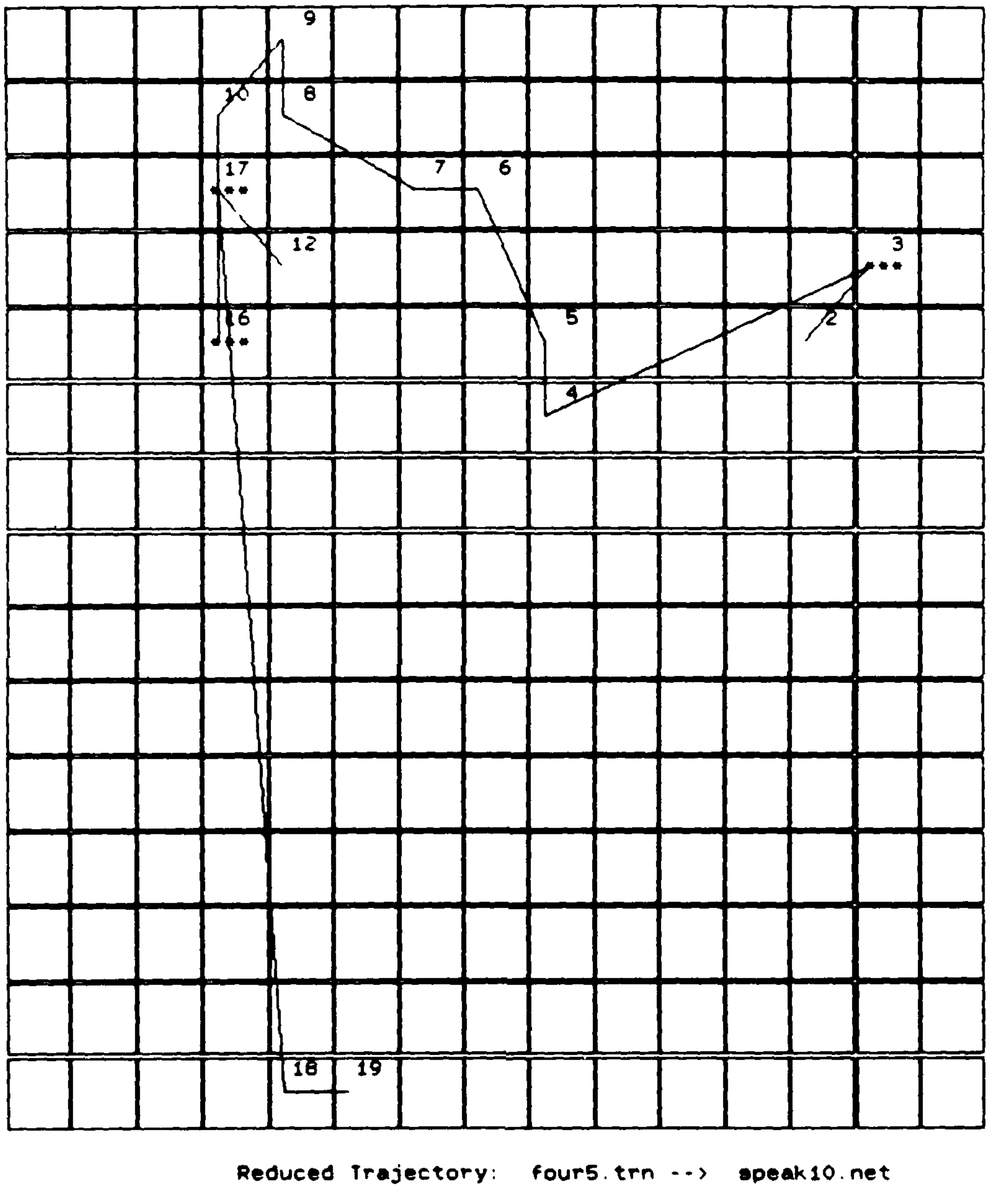
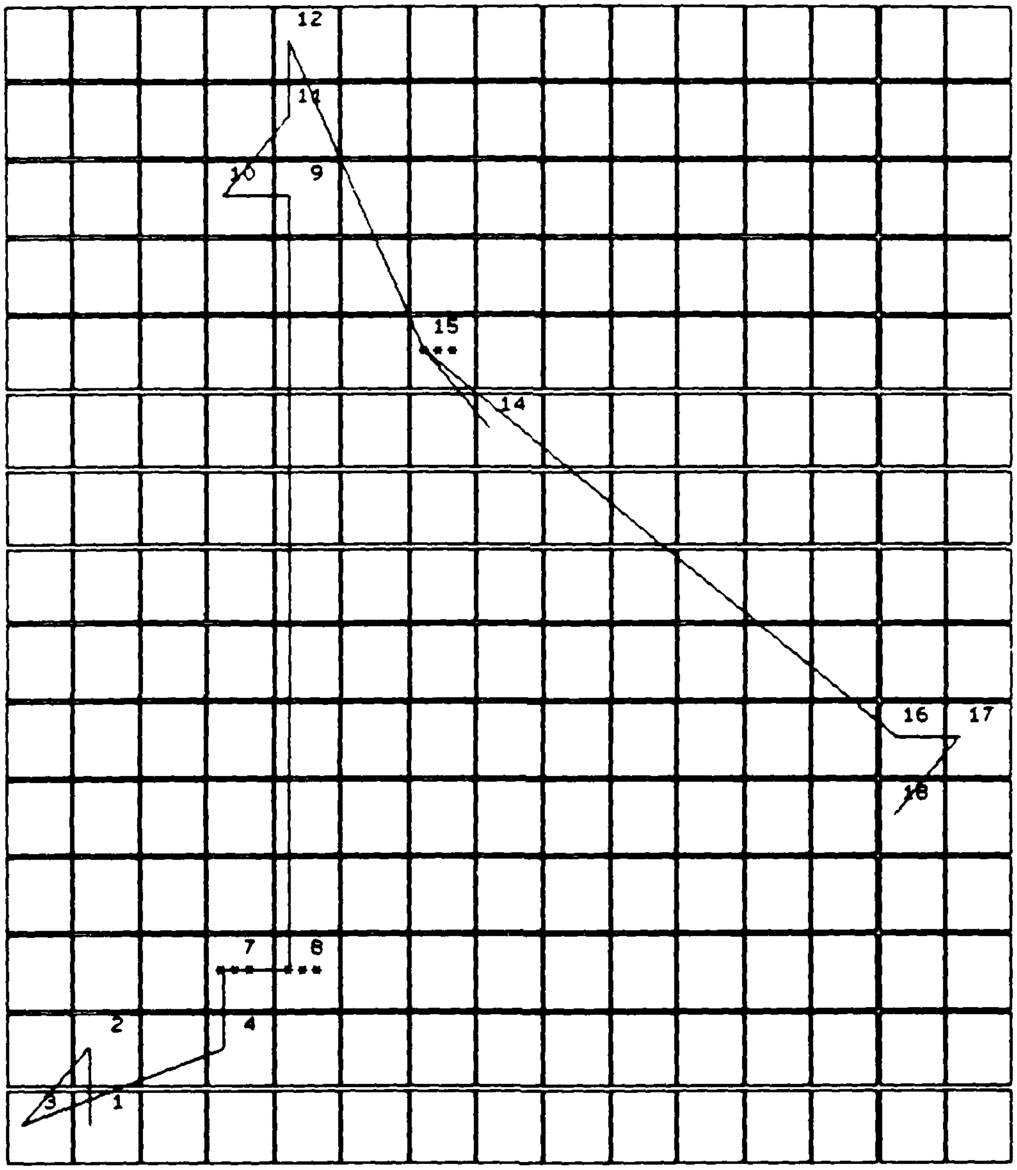


Figure A.5. Reduced Trajectory of the Template Four



Reduced Trajectory: five0.trn --> speak10.net

Figure A.6. Reduced Trajectory of the Template Five

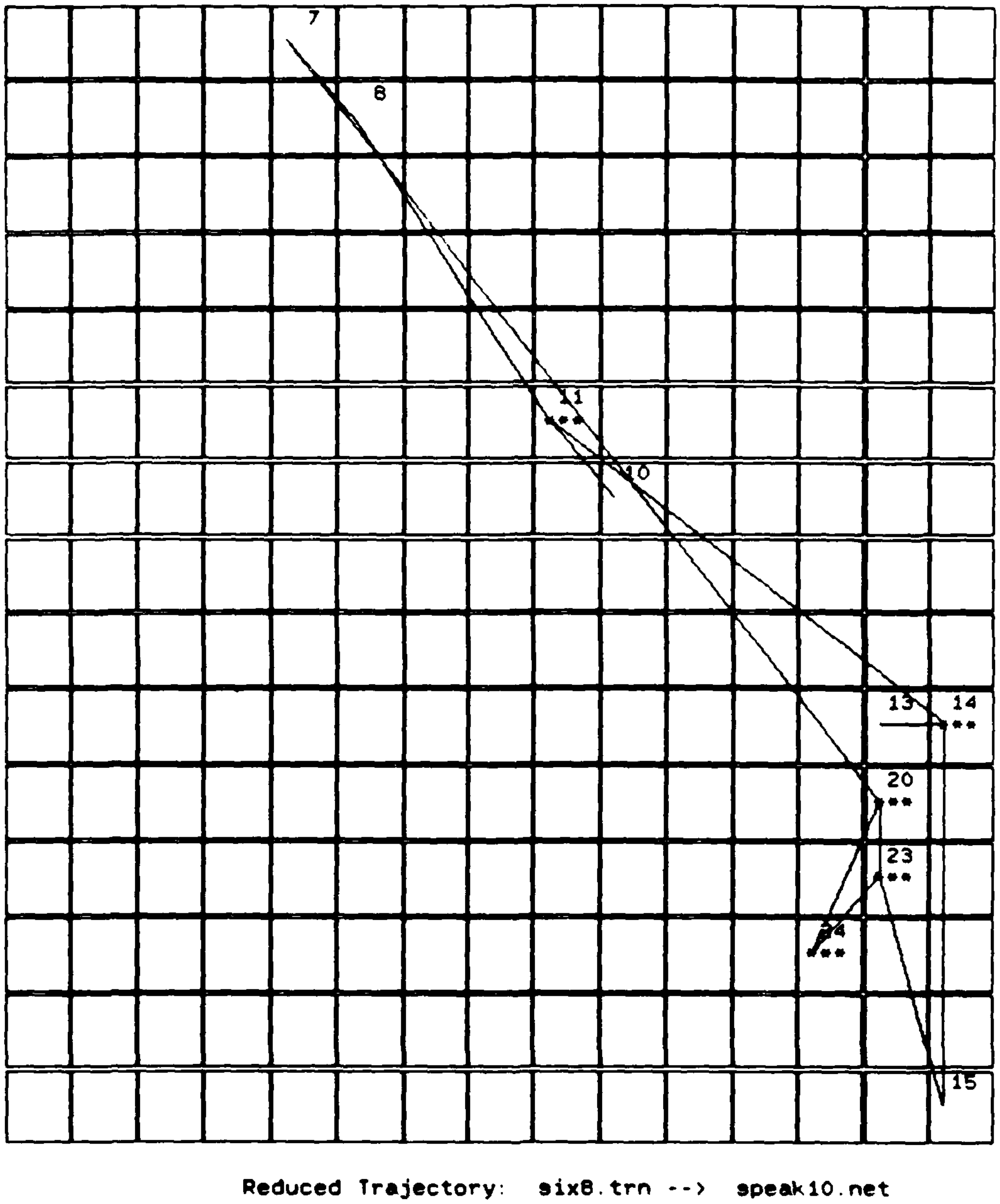
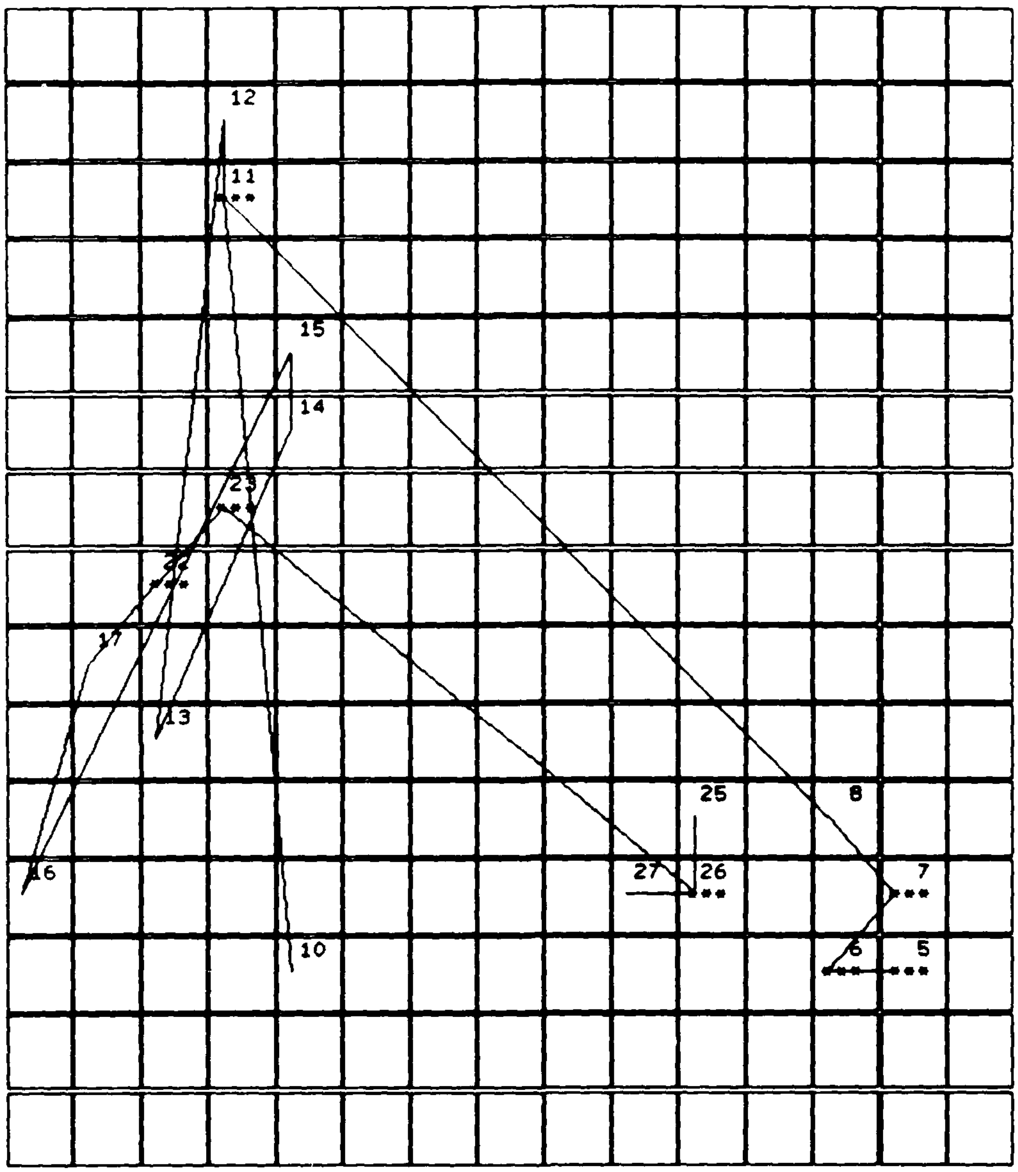
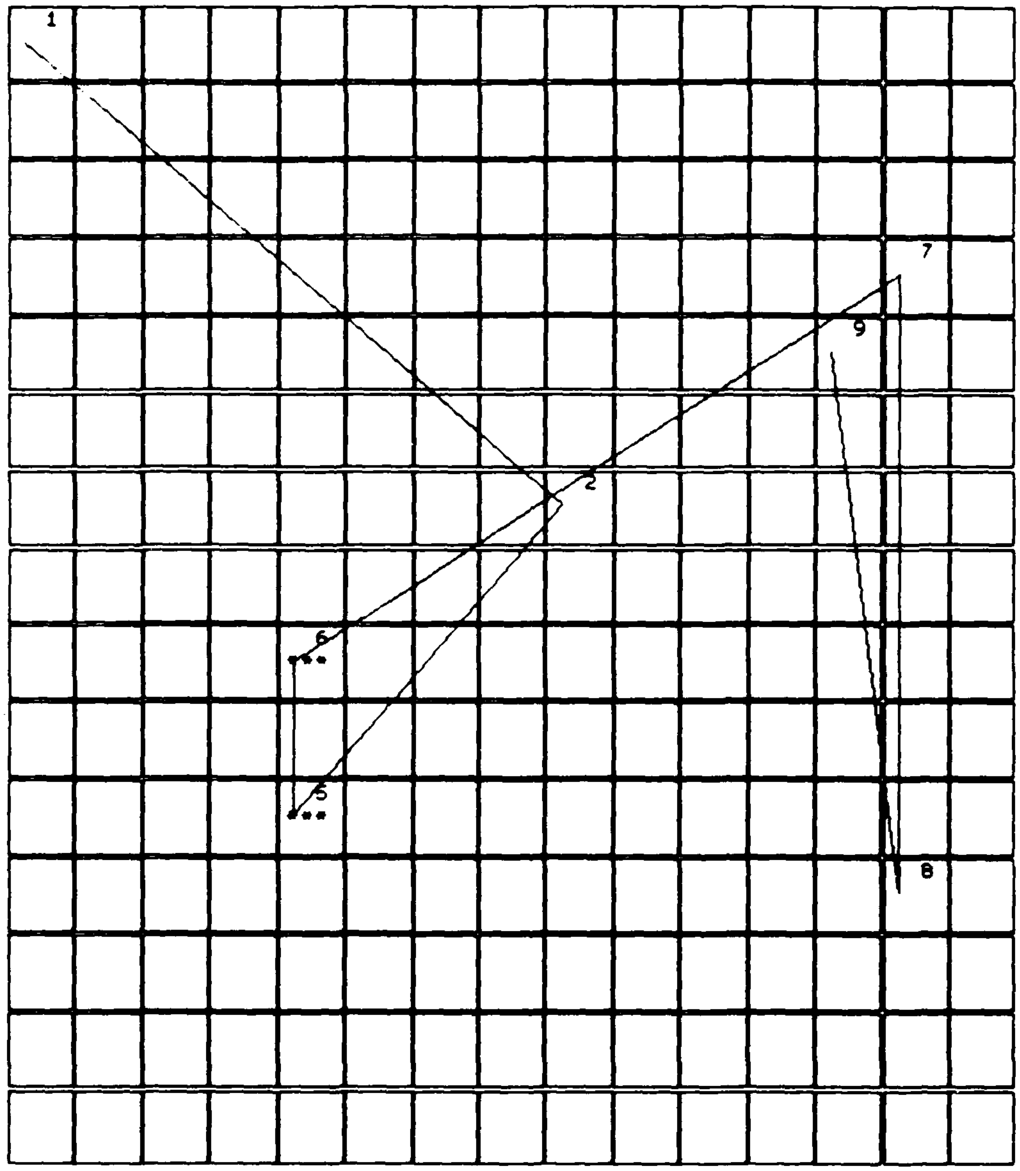


Figure A.7. Reduced Trajectory of the Template Six



Reduced Trajectory: seven0.trn --> speak10.net

Figure A.8. Reduced Trajectory of the Template Seven



Reduced Trajectory: eight0.trn --> speak10.net

Figure A.9. Reduced Trajectory of the Template Eight

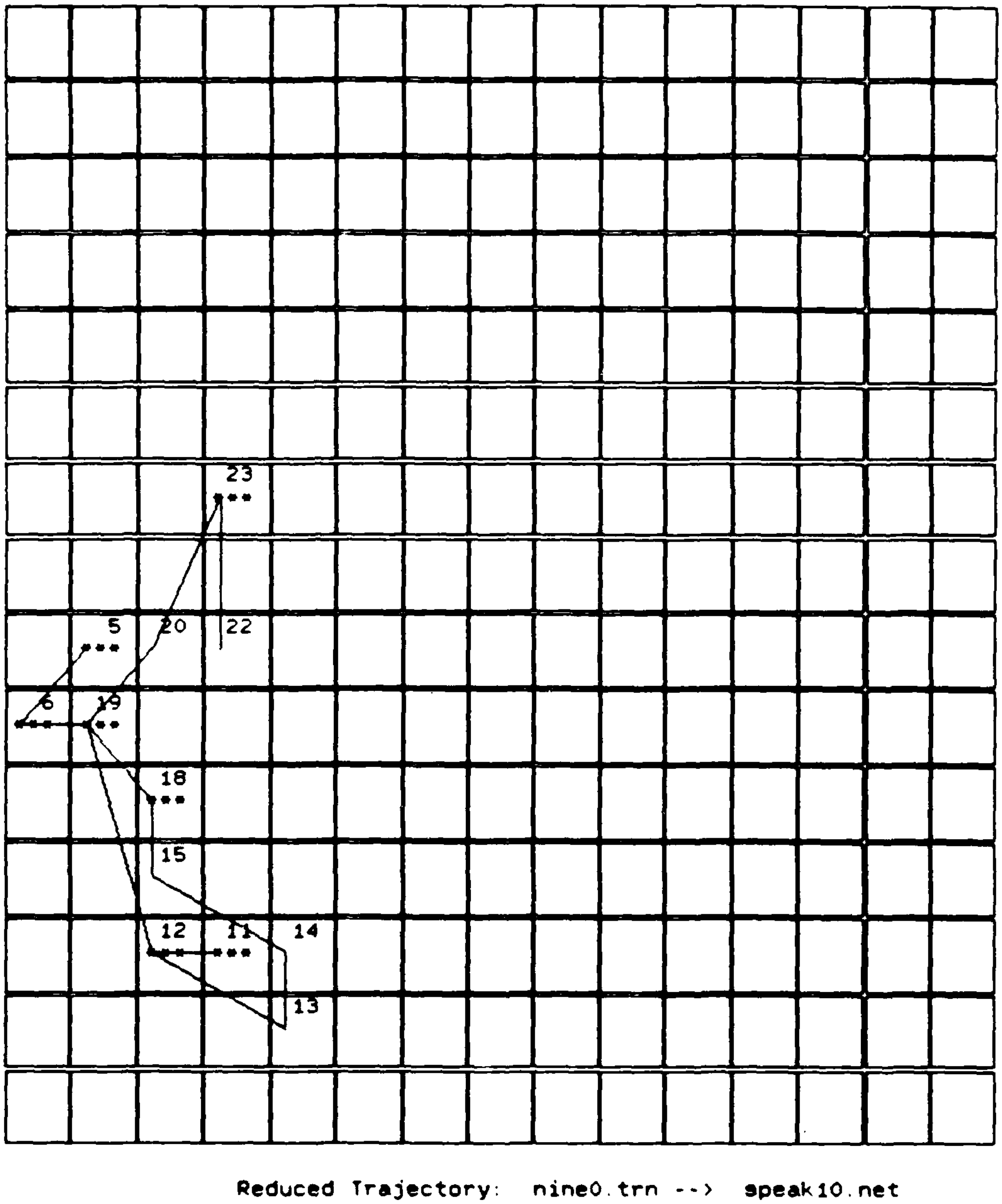
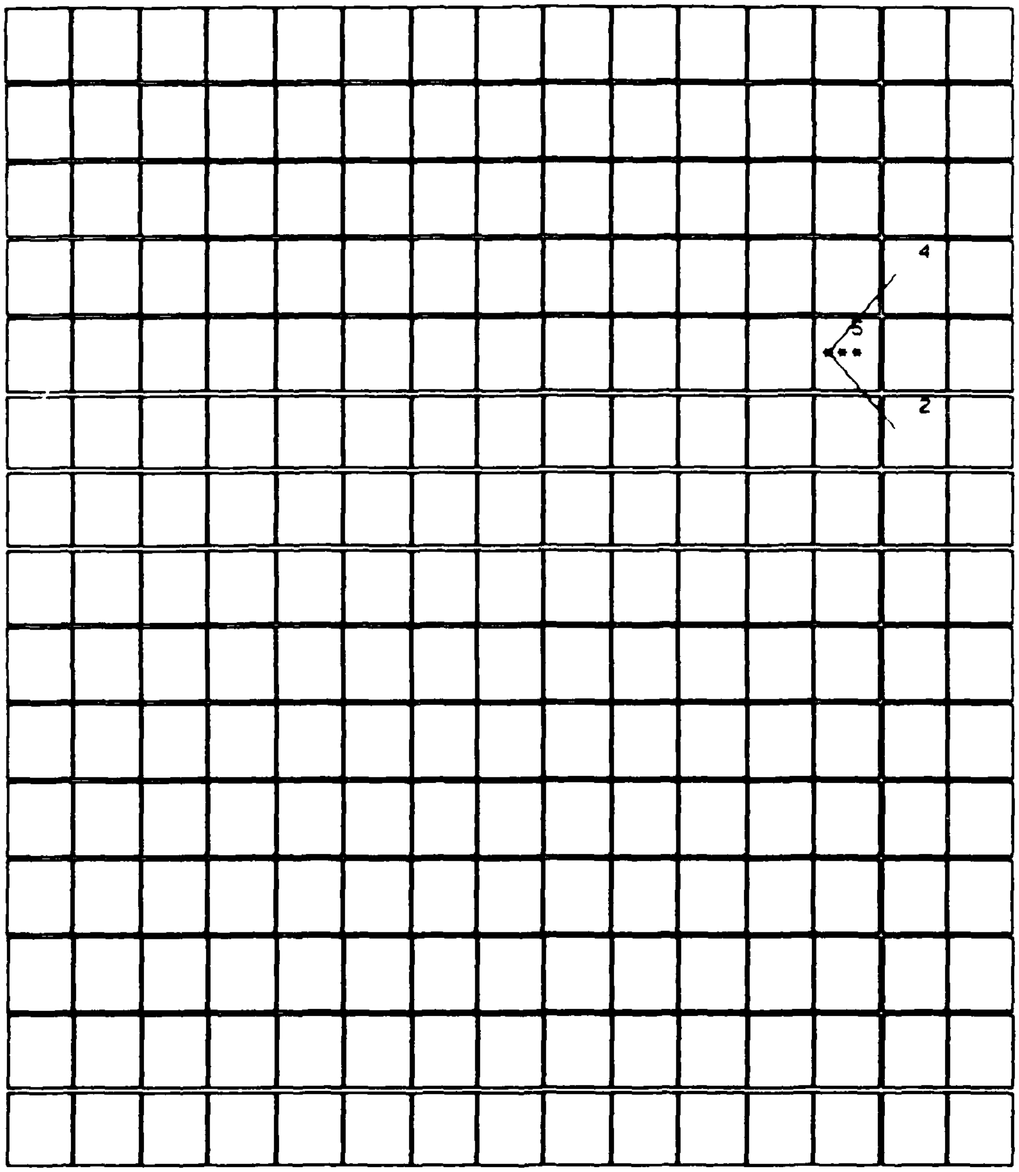


Figure A.10. Reduced Trajectory of the Template Nine



Reduced Trajectory: silence3.trn --> speak10.net

Figure A.11. Reduced Trajectory of the Template Silence

Appendix B: Computer Programs

There are a few points about this appendix that make it easier fc .he reader. Headers on each page show the appendix title on the left and the appropriate program title on the right. At the top of the first page of each program, the link command for that program is listed. In the link command are those *.c files used to create the program. Each program is shown below (in the sequence of its use in the body of this thesis) along with its page number and the files used to create it. Source files are only listed once, with the first program to use them.

Unfortunately, many of the programs and files use subroutines with the same names and either no or small differences. Because of the difficulty of identifying the differences, such subroutines are listed repeatedly. The exception to this is the trajectory reduction process (usually found in the subroutine read_word), which is only listed once. Its complete and final form is found in the file autodtw.c. Thereafter, it is abbreviated as '...Trajectory Reduction...'.

List of Programs and Files

```
$ link autofft,options_file/opt
/*
****************************** autofft.c ********************************

        INPUT: sound.hdr file
               *.snd files

        OUTPUT: *.trn files

        This routine takes the *.snd files named in sound.hdr and transforms
        them into *.trn files while leaving the original *.snd files
        unchanged. *.snd files must contain a leading eight bytes of
        header (thrown away) and an unspecified number of bytes of sampled
        sound data. The current assumption is that the sound is sampled
        logarithmically at 16 kHz with each sample being one byte.

        The transformation operates on 256 samples, moving forward in
        turn 85, 85 and 86 samples so that an overlap ratio of 3:1 is
        obtained. Each tranformation cycle takes the respective 256
        samples, multiplies by a Hamming window function and then does
        a 256 point FFT. The resulting 128 frequency magnitudes are
        reduced to 15 by a pseudo-logarithmic reduction in which the
        compression ratio is greatest at the higher frequencies. (The
        Kohonen reduction scheme is also allowed by the code, but it
        does not produce acceptable results since no filter is included.)
        The resulting 15 components are averaged and the average in turn
        is subtracted from each component. The resulting 15 components
        are then energy normalized to one.

        Each 15 component vector (corresponding to the initial 256
        samples) is then written to a *.trn file.
*/

# include stdio
# include math

# define PI3.1415926536

        float     ham[256] ;

main ()
{
        FILE      *fin, *fhdr, *fout ;
        float     xr[256] ;
        int       i, j, n, counter, c, limit, temp ;
        int       eof_flag ;
        int       pointer, overlap, noise_flag ;
        char      name_in[30], name_out[30], temp_name[30] ;
        int       sum, th_limit, i_snd, num_files ;
        int       reduction_flag ;

        printf ("AUTOFFT: Time/Frequency Conversion for Kohonen Net ...\n\n") ;

        printf ("Enter (0) logarithmic, or (1) Kohonen reduction: ") ;
        scanf ("%d", &reduction_flag) ;

        n = 256 ;
        setup_hamming (n) ;

        fhdr = fopen ("sounds.hdr", "r") ;
```

```
fscanf (fhdr, "%d", &num_files) ;
for (i_snd = 0 ; i_snd < num_files ; i_snd++) {
        fscanf (fhdr, "%s", temp_name) ;
        sprintf (name_in, "%s.snd", temp_name) ;
        fin = fopen (name_in, "rb") ;
        sprintf (name_out, "%s.trn", temp_name) ;
        fout = fopen (name_out, "w") ;
        printf (" %s opened ...", name_out) ;
        limit = 3000 ;
        th_limit = 0 ;

        counter = 0 ;
        overlap = 0 ;
        i = 0 ;
        pointer = 8 ;
        noise_flag = 0 ;
        sum = 0 ;
        eof_flag = 0 ;

        fseek (fin, pointer, 0) ;

        while (eof_flag != 1) {
                c = getc (fin) ;
                if (feof(fin) != 0)
                        eof_flag = 1 ;

/**** DATA is in the range [0,255] from the way the A/D
        software worked! ******/

                else {
                        if (c > 127)
                                c -= 256 ;
                        sum += abs (c) ;
                        xr[i++] = (float) c ;
                        }
                if ((i == n) && (eof_flag == 0)) {
                        hamming (n, xr) ;
                        ffter (n, xr) ;
                        if (reduction_flag == 0)
                                reduce_log (xr) ;
                        else
                                reduce_koh (xr) ;
                        subtract_ave (xr) ;
                        if (normalize (15, xr) != 0) {
                        if ((sum > th_limit) ||
                                        (noise_flag == 0)) {
                                        for (j = 0 ; j < 15 ; j++)
                                                fprintf (fout,
                                                "%f\n", xr[j]) ;
                                        if (++counter == limit)
                                                eof_flag = 1 ;
                                        }
                                else
                                        printf("\n Deleted %d @ %d\n",
                                                sum, counter) ;
                                }
                        i = 0 ;
                        if (overlap < 2) {
                                pointer += 85 ;
                                fseek (fin, pointer, 0) ;
```

```
                    overlap ++ ;
                    }
                else {
                    pointer += 86 ;
                    fseek (fin, pointer, 0) ;
                    overlap = 0 ;
                    }
                if (sum < th_limit)
                    noise_flag = 1 ;
                else
                    noise_flag = 0 ;
                sum = 0 ;
                }
            }
        fclose (fin) ;
        fclose (fout) ;
        printf (" %d vectors.\n", counter) ;
        }
    fclose (fhdr) ;
}

setup_hamming (n)
    int     n ;
{
    int     i ;

    /*****  Set up the lookup table (ham[i]) for the hamming window ***/

    for (i = 0 ; i < n ; i++)
        ham[i] = 0.54 - 0.46 * cos(2.0 * PI * i / (n - 1.0)) ;
}

hamming (n, xr)
    int     n ;
    float   xr[256] ;
{
    int     i ;

    for (i = 0 ; i < n ; i++)
        xr[i] *= ham[i] ;
}


ffter (n, xr)          /* FFT */
    int     n ;
    float   xr[256] ;
{
    int nv, nm, i, j, k, m, Le, Ld, p ;
    float   xi[256], ur, ui, rt, it, wr, wi, up ;

    for (i = 0 ; i < n ; i++)
        xi[i] = 0.0 ;

    nv = n / 2 ; nm = n - 1 ; j = 1 ;
    m = log((n+1) * 1.0) / log(2.0) ;

    for (i = 1 ; i <= n ; i++) {
        xr[n - i + 1] = xr[n - i] ;
        xi[n - i + 1] = xi[n - i] ;
        }
        for (i = 1 ; i < = nm ; i+ +) {
                if (i < j) {
                        rt = xr[j] ;
                        it = xi[j] ;
                        xr[j] = xr[i] ; xi[j] = xi[i] ;
                        xr[i] = rt ;
                        xi[i] = it ;
                        }
                k = nv ;
                while (k < j) {
                        j -= k ;
                        k /= 2 ;
                        }
                j + = k ;
                }
        for (k = 1 ; k < = m ; k+ +) {
                Ld = pow (2.0, k * 1.0) ;
                Le = Ld / 2 ;
                wr = cos (PI / Le) ;
                wi = -sin (PI / Le) ;
                ur = 1.0 ;
                ui = 0.0 ;

                for (j = 1 ; j < = Le ; j+ +) {
                        for (i = j ; i < = n ; i + = Ld) {
                                p = i + Le ;
                                rt = xr[p] * ur - xi[p] * ui ;
                                it = xr[p] * ui + xi[p] * ur ;
                                xr[p] = xr[i] - rt ;
                                xi[p] = xi[i] - it ;
                                xr[i] = xr[i] + rt ;
                                xi[i] = xi[i] + it ;
                                }
                        up = ur * wr - ui * wi ;
                        ui = ui * wr + ur * wi ;
                        ur = up ;
                        }
                }
        for (i = 1 ; i < = n ; i+ +) {
                xr[i - 1] = xr[i] ;
                xi[i - 1] = xi[i] ;
                }
        for (i = 0 ; i < n/2.0 ; i+ +)/* get magnitude */
                xr[i] = sqrt (xr[i] * xr[i] + xi[i] * xi[i]) ;
}

normalize (n, xr )
        int     n ;
        float   xr[256] ;
{
        int     i ;
        double  sum = 0 ;

        for (i = 0 ; i < n ; i+ +)
                sum + = xr[i] * xr[i] ;
        sum = sqrt (sum) ;
        if (sum = = 0.0) {
                printf (" An input vector found to be ZERO thrown away ...\n");
                return (0) ;
                }
        for (i = 0 ; i < n ; i++)
                xr[i] /= sum ;
        return (1) ;
}

reduce_log (xr)
        float   xr[256] ;
{
        int     i, j, counter ;

        xr[0] = xr[3] ;
        xr[1] = xr[4] + xr[5] ;
        counter = 6 ;
        for (j = 2 ; j < 13 ; j++) {
                xr[j] = 0.0 ;
                for (i = counter ; i < counter + j ; i++)
                        xr[j] += xr[i] ;
                counter += j ;
                }
        xr[13] = 0.0 ;
        for (i = 82 ; i < 102 ; i++)
                xr[13] += xr[i] ;
        xr[14] = 0.0 ;
        for (i = 102 ; i < 128 ; i++)
                xr[14] += xr[i] ;
}

reduce_koh (xr)
        float   xr[256] ;
{
        int     i, j, counter ;

        for (counter = 0 ; counter < 11 ; counter++) {
                xr[counter] = 0.0 ;
                for (i = counter * 6 + 1 ; i < counter * 6 + 7 ; i++) {
                        xr[counter] += xr[i] ;
                        }
                }
        for (counter = 0 ; counter < 5 ; counter++) {
                xr[counter + 11] = 0.0 ;
                for (i = 67 + counter * 11 ; i < 78 + counter * 11 ; i++) {
                        xr[counter + 11] += xr[i] ;
                        }
                }
        for (i = 122 ; i < 128 ; i++)
                xr[15] += xr[i] ;
}

subtract_ave (xr)
        float   xr[256] ;
{
        int     i ;
        double  sum ;

        sum = 0.0 ;
        for (i = 0 ; i < 15 ; i++)
                sum += xr[i] ;
        sum /= 15.0 ;
        for (i = 0 ; i < 15 ; i++)
```

```
                xr[i] -= sum ;
}
```

```
$ link neural7,nweight4,options_file/opt
/*
******************************** neural7.c ********************************

        Routines to generate the first layer Kohonen network without
        graphics. This version includes CONSCIENCE -- the ability to
        temporarilly remove nodes from consideration for training when
        they have already received more than their share of training.

        Note that this version of conscience will only eliminate a node
        from being the "closest" to the input. It will still be trained
        if it is in the region of a node which is chosen as the "closest".
        ************************************************************

        Implementation of Kohonen neural network algorithm as illustrated and
        described in IEEE magazine, Apr 87, by Dr. Lippman.

                        Capt Gary Barmore, 25 Aug 88

        GENERAL:
        (1) Output nodes are stored in a m x n matrix with each node
        represented by weights associated with each of the input
        nodes. (Limited to 20 by 20 array)
        (2) Output nodes are initialized with values between [-0.05,+0.05].
        (3) For each iteration, inputs are taken from a *.trn file which
        contains a sequence of 15 component vectors generated by
        AUTOFFT.EXE
        (4) Gain curves may be either linear, sigmoidal (not very successful),
        or piecewise (two) linear.
        (5) The size of the neighborhood is reduced as a function of the
        percentage of loop completion. In piecewise linear gain runs,
        the second "piece" is hardwired to have neighborhoods constant
        at "1 1"; i.e. the closest node and its nearest neighbors (in
        a rectangular region).
*/

# include math
# include stdio
# include time

        int     conscience[20][20] ;/* records # times closest */
        int     nodes ;                         /* number of nodes */
        double  consc = 1.5 ;           /* conscience factor */
        long    its ;
        long    nodes_elim ;

        float   map[20][20][16] ;  /* output nodes */
        double  input[16] ;             /* input nodes */
        double  gain ;
        double  mcount ;
        double  percent ;

        int     closest[2] ;            /* closest node */
        int     neigh[2] ;              /* neighbor */
        int     nrangex, nrangey ;/* neighbor range */
        int     nfactorx, nfactory ; /* neighbor factor */
        long    count ;                         /* # of iterations */
        int     graph ;                         /* # between plots */
        int     seed ;
        int     maxneighx, maxneighy ;/* Starting area */
        int         minneighx, minneighy ;/* Final area */
        int         xsize, ysize ;                   /* Size of array */
        int         number_inputs ;

        char        training_file[30], net_file[30] ;
        char        net_name[15] ;

        struct curve {
                    int                 type ;
                    double              maxgain ;
                    double              mingain ;
                    double              midgain ;
                    int                 midtime ;
                    } gcurve ;

        struct flg {
                    int                 rnd_in ;
                    } flag ;

init (map)
        float       map[20][20][16] ;
{
        int         r, c, i ;
        float       max_rand = pow(2.0, 31.0) - 1.0 ;

        nodes_elim = 0 ;
        nodes = ysize * xsize ;
        for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                                conscience[r][c] = 0 ;
                                for (i = 0 ; i < number_inputs ; i++) {
                                            map[r][c][i] = rand () / max_rand / 10.0 - .05 ;
                                            }
                                }
                    }
}

mindist (map, inp, close)
        double              inp[16] ;
        int                 close[2] ;
        float               map[20][20][16] ;
{
        int                 r, c, i ;
        double              dot_product ;
        double              maximum = 0.0 ;

        for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                                if (conscience[r][c] < consc * its / nodes ) {
                                            dot_product = 0.0 ;
                                            for (i = 0 ; i < number_inputs ; i++)
                                                        dot_product += inp[i] * map[r][c][i] ;
                                            if (dot_product > maximum) {
                                                        maximum = dot_product ;
                                                        close[0] = c ;
                                                        close[1] = r ;
                                                        }
                                            }
                                else
                                            nodes_elim++ ;
```

```
                        }
                }
        conscience[close[1]][close[0]] += 1 ;
}

userinp ()
{
        int                     line ;
        int                     c ;
        struct tm               *localtime(), *time ;
        int                     *bintim ;

        do {
                printf ("NEURAL7 (Net training with conscience only!) ... \n\n") ;

                printf("Enter size 'm n' (for an m x n) of array = ? [int int] ") ;
                scanf("%d %d", &ysize, &xsize) ;
                if (ysize < 2)
                        ysize = 2 ;
                else if (ysize > 20)
                        ysize = 20 ;
                if (xsize < 2)
                        xsize = 2 ;
                else if (xsize > 20)
                        xsize = 20 ;

                printf("Enter name of training file [.trn assumed]: ") ;
                scanf ("%s", net_name) ;
                sprintf (training_file, "%s.trn", net_name) ;
                printf(" Training file is: %s\n", training_file) ;

                number_inputs = 15 ;
                if (number_inputs < 2)
                        number_inputs = 2 ;
                else if (number_inputs > 16)
                        number_inputs = 16 ;

                printf("Enter name of net file to create [.net appended]: ") ;
                scanf ("%s", net_name) ;
                sprintf (net_file, "%s.net", net_name) ;
                printf(" Net file to be created: %s\n", net_file) ;

                printf ("Number of iterations = ? [int] ") ;
                scanf ("%ld", &count) ;
                if (count <= 10 || count > 130000)
                        count = 100 ;
                mcount = (double) count ;

                printf ("Number of iterations between status messages = ? [int] ") ;
                scanf ("%d", &graph) ;
                if (graph < 1 || graph > count)
                        graph = 10 ;

                ingain () ;

                printf ("Do you want 0) sequential or 1) randomizedtraining? ") ;
                scanf ("%d", &flag.rnd_in) ;

                printf ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
                scanf ("%d %d", &maxneighy, &maxneighx) ;
        if (maxneighx < 2 || maxneighx > xsize - 1)
            maxneighx = 2 ;
        if (maxneighy < 2 || maxneighy > ysize - 1)
            maxneighy = 2 ;

        printf ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
        scanf ("%d %d", &minneighy, &minneighx) ;
        if (minneighx < 1 || minneighx > maxneighx)
            minneighx = 1 ;
        if (minneighy < 1 || minneighy > maxneighy)
            minneighy = 1 ;

        printf("Initial seed for random # generator (0 SELECTS TIME) = ? [int] ");
        scanf ("%d", &seed) ;
        if (seed == 0) {
            time = localtime (bintim) ;
            time.tm_sec %= 60 ;
            time.tm_min %= 60 ;
            seed = time.tm_sec * time.tm_min ;
            }
        srand (seed) ;

        printf("Ready to begin? (y/n) ") ;
        while ((c = getc (stdin)) == ' ' || c == '\n' || c == '\t')
            ;
        } while (c != 'y') ;
}

ingain ()
{
    int             line ;

    printf("For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR : ");
    scanf ("%d", &gcurve.type) ;

    if (gcurve.type == 0 || gcurve.type == 1) {
        printf ("Maximum gain = ? [float] ") ;
        scanf ("%E", &gcurve.maxgain) ;
        if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
            gcurve.maxgain = .99 ;

        printf ("Minimum gain = ? [float] ") ;
        scanf ("%E", &gcurve.mingain) ;
        if (gcurve.mingain <= 0.0 || gcurve.mingain >= 1.0)
            gcurve.mingain = 0.0 ;

        }
    else {
        printf ("First segment starting gain = ? [float] ") ;
        scanf ("%E", &gcurve.maxgain) ;
        if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
            gcurve.maxgain = .99 ;

        printf ("Second segment starting gain = ? [float] ") ;
        scanf ("%E", &gcurve.midgain) ;
        if (gcurve.midgain <= 0.0 || gcurve.midgain >= 1.0)
            gcurve.midgain = 0.0 ;

        printf ("Second segment starting iteration = ? [float] ") ;
        scanf ("%d", &gcurve.midtime) ;
```

```
                if (gcurve.midtime < = 0 | | gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long    i ;
{
        if (gcurve.type = = 0)
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) + gcurve.mingain ;
        else if (gcurve.type = = 1)
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i - count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
                }
}

save_net ()
{
        int             r, c, i ;
        FILE            *fnet ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
        for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long            i ;
        char            s1[10] ;
        int             k ;
        int             ws_id = 1 ;
        int             clear_flag = 1;
        FILE            *tf ;
        extern unsigned _stklen ;

        _stklen = 8192 ;

        userinp () ; /* Get input values */

        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */

        read_trn_file () ;

        for (i = 1 ; i < = count ; i++) {
                its = i ;
                if (i % graph == 0) {
                        printf ("NEURAL3: gain = %f, yrange = %d, ", gain, nrangey) ;
                        printf ("xrange = %d, iteration # %d", nrangex,i) ;
                        printf (" (of %ld)\n", count) ;
                        k = nodes_elim / (double) graph ;
                        printf ("%d ave nodes eliminated!\n", k) ;
                        nodes_elim = 0 ;
                        }
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in == 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (map, input, closest) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
********************************* nweight4.c *********************************

        These routines allow training and testing of a first layer Kohonen
        network. The training primarilly supports NEURAL7.C/EXE. Testing
        (with true sound spectrum data) supports NEURAL2.C/EXE to recreate
        sounds.

*/

# include math
# include stdio
# include stat

        extern double           input[16] ;             /* input nodes */
        extern double           gain ;

        extern int              closest[2] ;            /* closest node */
        extern int              neigh[2] ;              /* neighbor */
        extern int              xsize, ysize ;          /* Size of array */
        extern int              number_inputs ;
        extern char             training_file[30] ;
        int                 tr_length ;
        float               tr_data[22500] ;
        int                 tr_counter = 0 ;
        int                 tr_vectors ;
        int                 node_sound[225] ;
        int                 num_words ;
        int                 word_limits[25][2] ;

read_trn_file ()
{
        FILE                *tf ;
        float               value = 1.0 ;
        unsigned            memory ;

        tf = fopen (training_file, “r”) ;
        tr_length = 0 ;
        while (feof(tf) = = 0) {
                fscanf (tf, “%f”, &value) ;
                *(tr_data + tr_length) = value ;
                tr_length+ + ;
                }
        fclose (tf) ;
        tr_length-- ;
        tr_vectors = floor (tr_length / 15.0) ;
        tr_length = 15 * tr_vectors ;
}

getin ()
{
        int                 i ;

        if (tr_counter = = tr_length)
                tr_counter = 0 ;
        for (i = 0 ; i < 15 ; i+ +) {
                input[i] = *(tr_data + tr_counter) ;
                tr_counter+ + ;
                }
}

get_rnd_in ()
{
        int                 i ;
        double              max_rand = pow (2.0, 31.0) - 1.0 ;
        int                 pointer ;

        pointer = 15 * floor ((rand() * (tr_vectors - .0001) / max_rand)) ;
        for (i = 0 ; i < 15 ; i+ +)
                input[i] = *(tr_data + pointer + i) ;
}

weightem (map)
        float               map[20][20][16] ;
{
        int                 nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
                if (nright > = xsize)
                        nright = xsize - 1 ;
        nleft = closest[0] - neigh[0] + 1 ;
        if (nleft < 0)
            nleft = 0 ;
        nup = closest[1] - neigh[1] + 1 ;
        if (nup < 0)
            nup = 0 ;
        ndown = closest[1] + neigh[1] - 1 ;
        if (ndown > = ysize)
            ndown = ysize - 1 ;

        }
    else {
        nright = closest[0] ;
        nleft = closest[0] ;
        nup = closest[1] ;
        ndown = closest[1] ;
        }

    for (r = nup; r < = ndown ; r+ +) {
        for (c = nleft ; c < = nright ; c+ +) {
            for (i = 0 ; i < number_inputs ; i+ +)
                map[r][c][i] + = gain * (input[i] - map[r][c][i]) ;
            }
        }
}
```

```
$ link neural2,nplot,nprinter,mat2,nweight4,options_file/opt
/*
******************************** neural2.c ********************************

          Routines to train first Kohonen neural net (with graphics) and
          to display spectra of net after it is trained. During training,
          the graphics show spectra. However, this slows down the training
          greatly. Thus if time is important, run NEURAL7.EXE.
          *****************************************************

          Implementation of Kohonen neural network algorithm as illustrated and
          described in IEEE magazine, Apr 87, by Dr. Lippman.

                                        Capt Gary Barmore, 7 Feb 88

          GENERAL:
          (1) Output nodes are stored in a m x n matrix with each node
          represented by weights associated with each of the input
          nodes. (Limited to 20 by 20 arrays).
          (2) Output nodes are initialized with values between [-0.05,+0.05].
          (3) For each iteration, input nodes receive values consisting of
          15 component vectors taken from a *.trn file generated by
          AUTOFFT.EXE.
          (4) Gain curves may be linear, sigmoidal (not very successful) or
          piecewise (two pieces only) linear.
          (5) The size of the neighborhood is reduced as a function of the
          percentage of loop completion. For piecewise linear gain runs,
          the second "piece" is hardwired for a neighborhood of "1 1"; i.e.
          it includes the closest node and its nearest neighbors in a
          rectangular grid.
*/

# include math
# include stdio
# include curses
# include time
# include <gksdefs.h>

# define bool      int

          float     map[20][20][16] ; /* output nodes */
          double    input[16] ; /* input nodes */
          double    gain, noise ;
          double    mcount ;
          double    percent ;

          int       closest[2] ; /* closest node */
          int       neigh[2] ; /* neighbor */
          int       nrangex, nrangey ; /* neighbor range */
          int       nfactorx, nfactory ; /* neighbor factor */
          long      count ; /* # of iterations */
          int       graph ; /* # between plots */
          int       seed ;
          int       maxneighx, maxneighy ; /* Starting area */
          int       minneighx, minneighy ; /* Final area */
          int       xsize, ysize ; /* Size of array */
          int       number_inputs ;

          char      net_file[30], net_name[15] ;
          char      training_file[30] ;
```

```
        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        struct flg {
                int             rnd_in ;
                } flag ;

init (map)
        float   map[20][20][16] ;
{
        int     r, c, i ;
        float   max_rand = pow(2.0, 31.0) - 1.0 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                map[r][c][i] = rand () / max_rand / 10.0 - .05;
                                }
                        }
                }
}

mindist (map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           map[20][20][16] ;
{
        int             r, c, i ;
        double          dot_product ;
        double          maximum = 0.0 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dot_product = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dot_product += inp[i] * map[r][c][i] ;
                        if (dot_product > maximum) {
                                maximum = dot_product ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}

userinp ()
{
        int             line ;
        int             c ;
        struct tm       *localtime(), *time ;
        int             *bintim ;

        do {
                initscr () ;
```

```
clear () ;
printw ("NEURAL2 (Training a Sound Net with GRAPHICS only!) ... \n\n") ;

printw("\n\nEnter size 'm n' (for an m x n) of array = ? [int int] ") ;
scanw("%d %d", &ysize, &xsize) ;
if (ysize < 2)
        ysize = 2 ;
else if (ysize > 20)
        ysize = 20 ;
if (xsize < 2)
        xsize = 2 ;
else if (xsize > 20)
        xsize = 20 ;

printw("Enter name of training file [less .trn]: ") ;
scanw ("%s", net_name) ;
sprintf (training_file, "%s.trn", net_name) ;
printw (" Training file is : %s\n", training_file) ;

number_inputs = 15 ;
if (number_inputs < 2)
        number_inputs = 2 ;
else if (number_inputs > 16)
        number_inputs = 16 ;

printw("Enter name of net file to create [less .net]: ") ;
scanw ("%s", net_name) ;
sprintf (net_file, "%s.net", net_name) ;
printw (" Net file to be created: %s\n", net_file) ;

printw ("Number of iterations = ? [int] ") ;
scanw ("%ld", &count) ;
if (count <= 10 || count > 130000)
        count = 100 ;
mcount = (double) count ;

printw ("Number of iterations between plots = ? [int] ") ;
scanw ("%d", &graph) ;
if (graph < 1 || graph > count)
        graph = 10 ;

ingain () ;

printw ("Do you want (0) sequential or (1) random training? ") ;
scanw ("%d", &flag.rnd_in) ;

printw ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
scanw ("%d %d", &maxneighy, &maxneighx) ;
if (maxneighx < 2 || maxneighx > xsize - 1)
        maxneighx = 2 ;
if (maxneighy < 2 || maxneighy > ysize - 1)
        maxneighy = 2 ;

printw ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
scanw ("%d %d", &minneighy, &minneighx) ;
if (minneighx < 1 || minneighx > maxneighx)
        minneighx = 1 ;
if (minneighy < 1 || minneighy > maxneighy)
        minneighy = 1 ;
```

```
                printw
                        ("Initial seed for random # generator (0 SELECTS TIME) = ? [int] ");
                scanw ("%d", &seed) ;
                if (seed = = 0) {
                        time = localtime (bintim) ;
                        time.tm_sec %= 60 ;
                        time.tm_min %= 60 ;
                        seed = time.tm_sec * time.tm_min ;
                        }
                srand (seed) ;

                printw("Ready to begin? (y/n) ") ;
                while ((c = getch ()) = = ' ' || c = = '\n' || c = = '\t')
                        ;
                endwin () ;
                } while (c != 'y') ;
}

ingain ()
{
        int     line ;

        printw("For gain enter 0) LINEAR, 1) SIGMOIDAL  2) PIECEWISE LINEAR : ");
        scanw ("%d", &gcurve.type) ;

        if (gcurve.type = = 0 || gcurve.type = = 1) {
                printw ("Maximum gain = ? [float] ") ;
                scanw ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain < = 0.0)
                        gcurve.maxgain = .99 ;

                printw ("Minimum gain = ? [float] ") ;
                scanw ("%E", &gcurve.mingain) ;
                if (gcurve.mingain < = 0.0 || gcurve.mingain >= 1.0)
                        gcurve.mingain = 0.0 ;
                }
        else {
                printw ("First segment starting gain = ? [float] ") ;
                scanw ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain < = 0.0)
                        gcurve.maxgain = .99 ;

                printw ("Second segment starting gain = ? [float] ") ;
                scanw ("%E", &gcurve.midgain) ;
                if (gcurve.midgain < = 0.0 || gcurve.midgain >= 1.0)
                        gcurve.midgain = 0.0 ;

                printw ("Second segment starting iteration = ? [float] ") ;
                scanw ("%d", &gcurve.midtime) ;
                if (gcurve.midtime < = 0 || gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long    i ;
{
        if (gcurve.type = = 0)
```

```
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) +
                        gcurve.mingain ;
        else if (gcurve.type == 1)
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i -
                        count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
                }
}

main ()
{
        int     c ;

        printf ("\nNEURAL2 (Sound net Training with GRAPHICS only!) ...\n") ;
        printf ("\nDo you want to train a net? (y/n) ") ;
        while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
                ;
        if (c == 'y')
                train_net () ;

        printf ("\nDo you want to draw spectra of a net? (y/n) ") ;
        while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
                ;
        if (c == 'y')
                draw_net_spectra () ;
}

save_net ()
{
        int                             r, c, i ;
        char                            name[30] ;
        FILE                            *fnet ;

        fnet = fopen (net_file, "w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

save_temp_net()
{
        int     r, c, i ;
        FILE    *fnet ;
        char    temp_net[30] ;

        sprintf (temp_net, "%s.mid", net_name) ;
        fnet = fopen (temp_net, "w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet, " %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

draw_net_spectra ()
{
        int                     flag, r, c, i, j, k ;
        char                    name[30], s[10] ;
        double                  in[16] ;
        float                   element ;
        int                     loc[2] ;
        FILE                    *fsnd, *fnet ;
        int                     sound[] = {8,43,3,21,47,9,28,1} ;
        int                     sounds = 8 ;
        int                     fft3_flag ;

        printf ("\nEnter name of net-file to test: ") ;
        scanf ("%s", name) ;
        fnet = fopen (name, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %f", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
        printf ("Was this generated by (0) FFT2 or (1) FFT3 ? ") ;
        scanf ("%d", &fft3_flag) ;

        graph_test (name) ;
        if (fft3_flag != 1)
                draw_grid (ysize, xsize) ;
        draw_spectra (map, ysize, xsize) ;
        scanf ("%s",s) ;
        clipoff () ;
        graphoff () ;
}

train_net ()
{
        long                    i ;
        char                    s1[10] ;
        int                     ws_id = 1 ;
        int                     clear_flag = 1;
        FILE                    *tf ;

        extern unsigned         _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */
        read_trn_file () ;
        prep_graph () ;
        for (i = 1 ; i <= count ; i++) {
                if (i % graph == 0) {
                        fixcolors (map) ;
                        clear_viewport () ;
                        statusem (gain, nrangey, nrangex, i) ;
                        draw_spectra (map, ysize, xsize) ;
                        }
                if (i % 10000 == 0)
                        save_temp_net () ;
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in == 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (map, input, closest) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                        ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                        ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
        scanf ("%s",s1) ;
        end_graph () ;
        print_data (map) ;
        distance_histogram (map, ysize, xsize) ;
}

graph_test (name)
        char    name[30] ;
{
        char    title[79], labelx[79] ;
        float   xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
        float   yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
        int     points = 5 ;
        int     ws_id = 1 ;
        int     clear_flag = 1 ;
        short   length ;

        sprintf(title,"NEURAL2: Kohonen %d x %d Neural Net -- %s",
                ysize, xsize, name) ;
        graphon () ;
        gks$clear_ws (&ws_id, &clear_flag) ;
        gks$polyline (&points, xloc, yloc) ;
        prepcolmat (ysize, xsize) ;
        length = (short) strlen (title) ;
        outtitle (title, length) ;
        length = (short) strlen (labelx) ;
        clipon () ;
}

prep_graph ()
{
        char    title[79], labelx[79] ;
        float   xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
        float   yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
        int     points = 5 ;
        int     ws_id = 1 ;
        int     clear_flag = 1 ;
        int     box = 3 ;
        short   length ;

        sprintf(title,"NEURAL2: Kohonen %d x %d Neural Net", ysize, xsize) ;
        sprintf(labelx,
                "[%4.2f,%4.2f] Gain, [%d,%d : %d,%d] Neighbors, %d Iterations",
                gcurve.maxgain, gcurve.mingain, maxneighy, minneighy, maxneighx,
                minneighx, count) ;
        graphon () ;
        gks$create_seg (&box) ;
        gks$polyline (&points, xloc, yloc) ;
        gks$close_seg (&box) ;
        prepcolmat (ysize, xsize) ;
        length = (short) strlen (labelx) ;
        outlabelx (labelx, length) ;
        length = (short) strlen (title) ;
        outtitle (title, length) ;
        pickcolors () ;
        clipon () ;
}

end_graph ()
{
        clipoff () ;
        graphoff () ;
}

fixcolors (map)
        float           map[20][20][16] ;
{
        int             r, c, i ;
        int             max = 0 ;
        double          constant ;
        double          temp[20][20] ;

        extern int      colmat[20][20] ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        temp[r][c] = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                temp[r][c] += pow(map[r][c][i], 2.0) ;
                        if (temp[r][c] > max)
                                max = temp[r][c] ;
                        }
                }
        if (max != 0.0)
                constant = 14.999 / max ;
        else
                constant = 14.999 ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        colmat[r][c] = floor (temp[r][c] * constant) + 1 ;
                        if (colmat[r][c] > 15)
                                colmat[r][c] = 15 ;
                        }
                }
}

/*
******************************* nplot.c *********************************

        Routines to enable graphics for Micro-VAX II workstation using GKS
        for Kohonen network speech recognition.
*/

# include math
# include stdio
# include <gksdefs.h>
# include <descrip.h>

# define MAX_STRING 80

        double          xlow, xup, ylow, yup ;
        double          xdel, ydel ;
        float           x[100], y[100] ;

graphon ()
{
        int             ws_id = 1 ;
        int             error_status, category, inquire_okay ;
        int             dummy_int, def_mode, regen_flag, ws_type ;
        int             rx, ry, units ;
        float           x, y ;
        float           x0 = 0.0 ;
        float           x1 = 639.0 ;
        float           y0 = 0.0 ;
        float           y1 = 349.0 ;

        int             deferral_mode = GKS$K_ASAP ;
        int             regen_mode = GKS$K_IRG_SUPPRESSED ;

        struct dsc$descriptor           dummy_dsc ;
        char                            dummy_string [MAX_STRING] ;
        $DESCRIPTOR (error_file, "sys$error:") ;

        inquire_okay = 0 ;
        dummy_dsc.dsc$a_pointer = dummy_string ;
        dummy_dsc.dsc$w_length = (short) MAX_STRING ;

        gks$open_gks (&error_file) ;
        gks$inq_ws_category (&GKS$K_WSTYPE_DEFAULT, &error_status, &category);
/*
*       Make sure workstation type is valid.
*/
        if ((error_status != inquire_okay) ||
                ((category != GKS$K_WSCAT_OUTIN) &&
                (category != GKS$K_WSCAT_MO))) {
                printf ("The specified workstation type is invalid\n") ;
                printf ("Error status: %d\n", error_status) ;
                return ;
                }
        gks$open_ws (&ws_id, &GKS$K_CONID_DEFAULT, &GKS$K_WSTYPE_DEFAULT);
        gks$activate_ws (&ws_id) ;
/*
*       Make sure deferral mode and regeneration flag are properly set.
*/
        gks$set_defer_state (&ws_id, &deferral_mode, &regen_mode) ;
        gks$inq_ws_type (&ws_id, &error_status, &dummy_dsc, &ws_type,
                &dummy_int) ;
        gks$inq_def_defer_state (&ws_type, &error_status, &def_mode,
                &regen_flag) ;
        if (error_status != inquire_okay) {
                printf ("The deferral inquiry caused an error\n") ;
                printf ("Error status: %d\n", error_status) ;
                return ;
                }
/*
*       Set up viewport for drawing figures.
*/
        gks$set_window (&ws_id, &x0, &x1, &y0, &y1) ;
        gks$select_xform (&ws_id) ;
}

graphoff ()
{
        int     ws_id = 1 ;

        gks$update_ws (&ws_id, &GKS$K_PERFORM_FLAG) ;
        gks$deactivate_ws (&ws_id) ;
        gks$close_ws (&ws_id) ;
        gks$close_gks () ;
}

clipon ()
{
        int     ws2 = 2 ;
        float   x0 = 0.0 ;
        float   x1 = 560.0 ;
        float   y0 = 0.0 ;
        float   y1 = 289.0 ;
        float   vx0 = 0.183 ;
        float   vx1 = 0.881 ;
        float   vy0 = 0.1143 ;
        float   vy1 = 0.9400 ;
        int     on = 1 ;

        gks$set_window (&ws2, &x0, &x1, &y0, &y1) ;
        gks$set_viewport (&ws2, &vx0, &vx1, &vy0, &vy1) ;
        gks$select_xform (&ws2) ;
        gks$set_clipping (&on) ;
}

clear_viewport ()
{
        float   xloc[] = {0.0, 560.0, 560.0, 0.0, 0.0} ;
        float     yloc[] = {289.0, 289.0, 0.0, 0.0, 289.0} ;
        int       points = 5 ;
        int       solid = 1 ;
        int       background = 0 ;
        int       black = 1 ;

        gks$set_fill_color_index (&background) ;
        gks$set_fill_int_style (&solid) ;
        gks$fill_area (&points, xloc, yloc) ;
        gks$set_fill_color_index (&black) ;
}

clipoff ()
{
        int       off = 0 ;
        int       ws_id = 1 ;
        gks$select_xform (&ws_id) ;
        gks$set_clipping (&off) ;
}

outaxes ()
{
        int       xx, yy, points ;

        points = 2 ;
        y[0] = y[1] = 39 ; x[0] = 60 ; x[1] = 620 ;
        gks$polyline (&points, x, y) ;/* Plot horizontal axis */

        x[0] = x[1] = 60 ; y[0] = 329 ; y[1] = 40 ;
        gks$polyline (&points, x, y) ;/* Plot vertical axis */

        x[0] = 57 ; x[1] = 59 ;
        for (yy = 39 ; yy < = 329 ; yy + = 58) {/* Plot y ticks */
                  y[0] = y[1] = yy ;
                  gks$polyline (&points, x, y) ;
                  }
        y[0] = 38 ; y[1] = 36 ;
        for (xx = 60 ; xx < = 620 ; xx + = 56) {/* Plot x ticks */
                  x[0] = x[1] = xx ;
                  gks$polyline (&points, x, y) ;
                  }
}

outlimits (x2,d2,e2,x1,d1,e1) /* x-axis then y-axis */
        double              x1, d1, x2, d2 ;
        int                 e1, e2 ;
{
        char                s[5], ss[9] ;
        $DESCRIPTOR(s_dsc,s) ;
        $DESCRIPTOR(ss_dsc,ss) ;

        int                 ticks[3], loop, xx, yy, flag ;
        double              xypos[3], delta[3] ;
        double              e ;
        float               xloc, yloc ;

        ticks[0] = 5 ; ticks[1] = 10 ;

        e = e2 ;
        xlow = x2 * pow (10.0,e) ;
```

```
    xup = xlow + 10.0 * d2 * pow (10.0,e) ;
    e = e1
    ylow = x1 * pow (10.0,e) ;
    yup = ylow + 5.0 * d1 * pow (10.0,e) ;

    delta[0] = d1 ; delta[1] = d2 ;
    xypos[0] = x1 ; xypos[1] = x2 ;
    for (flag = 0 ; flag <= 1 ; flag++) {
        for (loop = 0 ; loop <= ticks[flag] ; loop++) {
            xx = 7 + loop * 7 ;
            if (flag) {
                sprintf (s,"%4.1f",xypos[flag]) ;
                xloc = 1.0 + 8.0 * (xx - 2.0) ;
                yloc = 34.0 ;
                gks$text (&xloc, &yloc, &s_dsc) ;
                }
            else {
                sprintf (s,"%4.1f",xypos[flag]) ;
                xloc = 22.0 ;
                yloc = 349.0 - 16.0 - (5.0 - loop) * 58.0 ;
                gks$text (&xloc, &yloc, &s_dsc) ;
                }
            xypos[flag] += delta[flag] ;
            }
        }
    if (e2 != 0) {
        sprintf (ss,"[x E%3d]",e2) ;
        xloc = 312.0 ;
        yloc = 22.0 ;
        gks$text (&xloc, &yloc, &ss_dsc) ;
        }
    if (e1 != 0) {
        xloc = -1.0 ;
        yloc = 0.0 ;
        gks$set_text_upvec (&xloc, &yloc) ; /* rotate text to 90 deg */
        sprintf (ss,"[x E%3d]",e1) ;
        xloc = 26.0 ;
        yloc = 209.0 ;
        gks$text (&xloc, &yloc, &ss_dsc) ;
        xloc = 0.0 ;
        yloc = 1.0 ;
        gks$set_text_upvec (&xloc, &yloc) ; /* change text to normal */
        }

    xdel = (xup - xlow) / 560.0 ;
    ydel = (yup - ylow) / 290.0 ;
}

outtitle (string, length)
    short           length ;
    char            *string ;
{
    int             title = 1 ;
    int             loc ;
    float           xloc, yloc ;
    struct dsc$descriptor string_dsc = {length,
                                        DSC$K_DTYPE_T,
                                        DSC$K_CLASS_S,
                                        string} ;
    xloc = 320.0 - 3.0 * length ;
```

```
        yloc = 343.0 ;
        gks$create_seg (&title) ;
        gks$text (&xloc, &yloc, &string_dsc) ;
        gks$close_seg (&title) ;
}

outlabelx (string, length)
        short               length ;
        char                *string ;
{
        int                 labelx = 2 ;
        int                 loc ;
        float               xloc, yloc ;
        struct dsc$descriptor string_dsc = {length,
                                            DSC$K_DTYPE_T,
                                            DSC$K_CLASS_S,
                                            string} ;

        xloc = 320.0 - 3.0 * length ;
        yloc = 10.0 ;
        gks$create_seg (&labelx) ;
        gks$text (&xloc, &yloc, &string_dsc) ;
        gks$close_seg (&labelx) ;
}

outlabely (string)
        char                string[80] ;
{
        int                 loc ;
        float               xloc, yloc ;
        $DESCRIPTOR(string_dsc,string) ;

        loc = strlen(string) ;
        string[loc] = '\0' ;
        xloc = -1.0 ;
        yloc = 0.0 ;
        gks$set_text_upvec (&xloc, &yloc) ;
        xloc = 13.0 ;
        yloc = 349.0 - 164.0 - 4.0 * loc ;
        gks$text (&xloc, &yloc, &string_dsc) ;
        xloc = 0.0 ;
        yloc = 1.0 ;
        gks$set_text_upvec (&xloc, &yloc) ;
}

/*
************************************ nprinter.c ******************************

        Simulates printer output by storing Kohonen training data in a file
        PRINTER.OUT. This file was written when NEURAL*.EXE was still
        being written and run on a Tandy 4000. The current version of the
        Kohonen training routines may not have the appropriate hooks in them
        to run these printer routines.
*/

# include stdio
# include math

        FILE                *fp, *fopen() ;
        double                  nei ;

        extern int              xsize, ysize, number_inputs, wrap_flag ;
        extern struct curve {
                int                     type ;
                double                  maxgain ;
                double                  mingain ;
                double                  midgain ;
                int                     midtime ;
                }       gcurve ;

popen ()
{
        fp = fopen ("printer.out","w") ;
        fputc (15,fp) ;
}

pclose ()
{
        fclose (fp) ;
}

pfeed ()
{
        fputc ('\n',fp) ;
}

p_return (lines)
        int     lines ;
{
        int     i ;

        for (i = 1 ; i <= lines ; i++)
                fprintf (fp,"\n") ;
}

print_data (map)
        float                   map[20][20][16] ;
{
        int                     r, c, i ;

        extern int              count, maxneighx, maxneighy, minneighx, minneighy ;
        extern int              seed ;
        extern double           xoff, yoff ;

        printf ("\nDo you want parameters and weights printed out? (y/n) ") ;
        while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
                ;
        if (c != 'y')
                return (0) ;

        popen () ;
        pfeed () ;

        fprintf(fp,"\nNEURAL...\n------\n\n") ;
        fprintf(fp,"SIZE OF ARRAY : %d x %d\n",ysize,xsize) ;
        fprintf(fp,"NUMBER OF INPUTS : %d\n",number_inputs) ;
        if (gcurve.type == 0) {
                fprintf(fp,"GAIN CURVE IS : %s\n","LINEAR") ;
                fprintf(fp,"RANGE OF GAIN : [%g,%g]\n",
                        gcurve.maxgain,gcurve.mingain) ;
                }
        else if (gcurve.type = = 1) {
                fprintf(fp,"GAIN CURVE IS : %s\n",
                        "SIGMOIDAL") ;
                fprintf(fp,"RANGE OF GAIN : [%g,%g]\n",
                        gcurve.maxgain,gcurve.mingain) ;
                }
        else {
                fprintf(fp,"GAIN CURVE IS : %s\n",
                        "PIECEWISE LINEAR") ;
                        fprintf(fp,"MAXIMUM GAINS (AND BREAKPOINT) : %g,%g (%d)\n",
                        gcurve.maxgain, gcurve.midgain, gcurve.midtime) ;
                }
        fprintf(fp,"NEIGHBORHOODS START AT : %d %d\n",
                maxneighy, maxneighx);
        fprintf(fp,"NEIGHBORHOODS END AT : %d %d\n",
                minneighy, minneighx);
        fprintf(fp,"SEED : %d\n",seed) ;
        fprintf(fp,"INITIAL X-OFFSET, Y-OFFSET : %g %g\n",xoff,yoff) ;
        if (wrap_flag = = 0)
                fprintf (fp,"WRAP : OFF\n") ;
        else
                fprintf (fp,"WRAP : ON\n") ;

        p_return (3) ;
        fprintf (fp,"Final values for weights are:\n\n") ;

        for (r = 0 ; r < ysize ; r++) {
                for (i = 0 ; i < number_inputs ; i++) {
                        for (c = 0 ; c < xsize ; c++)
                                fprintf (fp,"%5.1f ", map[r][c][i]) ;
                        p_return (1) ;
                        }
                p_return (1) ;
                }
        pclose () ;
}

distance_histogram (map, ysize, xsize)
        int                     ysize, xsize ;
        float                   map[20][20][16] ;
{
        int                     bin[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} ;
        int                     bin2[] = {0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0} ;
        float                   maximum = 0.0 ;
        float                   minimum = 9999.0 ;
        int                     binmax = 0 ;
        int                     binmax2= 0 ;
        float                   dist[20][20][4], a, b, delta ;
        int                     r, c, i, j ;
        double                  tot, sub ;

        printf ("\nDo you want distance histogram printed out? (y/n) ") ;
        while ((c = getchar()) = = ' ' || c = = '\n' || c = = '\t')
                ;
        if (c != 'y')
                return (0) ;

        popen () ;
```

```
pfeed () ;
fprintf (fp,"The distance histogram is:\n\n") ;

for (r = 0 ; r < ysize ; r++) {
    for (c = 0 ; c < xsize ; c++) {
        if (r == 0)
            dist[r][c][0] = -1.0 ;
        else {
            dist[r][c][0] = 0.0 ;
            for (i = 0 ; i < number_inputs ; i++)
                dist[r][c][0] +=
                pow(map[r][c][i]-map[r-1][c][i],2.0);
            dist[r][c][0] = sqrt ((double) dist[r][c][0]) ;
            test_maxmin (dist[r][c][0], &maximum, &minimum);
            }
        if (r == ysize - 1)
            dist[r][c][2] = -1.0 ;
        else {
            dist[r][c][2] = 0.0 ;
            for (i = 0 ; i < number_inputs ; i++)
                dist[r][c][2] +=
                pow(map[r][c][i]-map[r+1][c][i],2.0);
            dist[r][c][2] = sqrt ((double) dist[r][c][2]) ;
            test_maxmin (dist[r][c][2], &maximum, &minimum);
            }
        if (c == 0)
            dist[r][c][1] = -1.0 ;
        else {
            dist[r][c][1] = 0.0 ;
            for (i = 0 ; i < number_inputs ; i++)
                dist[r][c][1] +=
                pow(map[r][c][i]-map[r][c-1][i],2.0);
            dist[r][c][1] = sqrt ((double) dist[r][c][1]) ;
            test_maxmin (dist[r][c][1], &maximum, &minimum);
            }
        if (c == xsize - 1)
            dist[r][c][3] = -1.0 ;
        else {
            dist[r][c][3] = 0.0 ;
            for (i = 0 ; i < number_inputs ; i++)
                dist[r][c][3] +=
                pow(map[r][c][i]-map[r][c+1][i],2.0);
            dist[r][c][3] = sqrt ((double) dist[r][c][3]) ;
            test_maxmin (dist[r][c][3], &maximum, &minimum);
            }
        }
    }
tot = 0.0 ;
sub = 0.0 ;
delta = 5.0 * sqrt ((double) number_inputs) / 20.0 ;
for (j = 0 ; j < 20 ; j++) {
    a = j * delta ;
    b = (j + 1.0) * delta ;
    for (r = 0 ; r < ysize ; r++) {
        for (c = 0 ; c < xsize ; c++) {
            for (i = 0 ; i < 4 ; i++) {
                if ((dist[r][c][i] <= b) &&
                (dist[r][c][i] >= a)) {
                    bin2[j]++ ;
                    tot += 1.0 ;
```

```
                                                }
                                        }
                                }
                        }
                if (bin2[j] > binmax2)
                        binmax2 = bin2[j] ;
                }
        for (j = 0 ; j < 3 ; j++)
                sub += bin2[j] ;
        nei = sub / tot ;

        for (j = 0 ; j < 20 ; j++) {
                a = j * delta ;
                b = (j + 1.0) * delta ;
                fprintf (fp,"[%6.2f -> %6.2f] %4d ", a, b, bin2[j]) ;
                for (i = 0 ; i < 40 * bin2[j] / binmax2 ; i++)
                        fputc ('*',fp) ;
                p_return (1) ;
                }
        p_return (2) ;
        delta = (maximum - minimum) / 10.0 ;
        for (j = 0 ; j < 10 ; j++) {
                a = minimum + j * delta ;
                b = minimum + (j + 1.0) * delta ;
                for (r = 0 ; r < ysize ; r++) {
                        for (c = 0 ; c < xsize ; c++) {
                                for (i = 0 ; i < 4 ; i++) {
                                        if ((dist[r][c][i] <= b) &&
                                        (dist[r][c][i] >= a))
                                                bin[j]++ ;
                                        }
                                }
                        }
                if (bin[j] > binmax)
                        binmax = bin[j] ;
                }
        for (j = 0 ; j < 10 ; j++) {
                a = minimum + j * delta ;
                b = minimum + (j + 1.0) * delta ;
                fprintf (fp,"[%6.2f -> %6.2f] %4d ", a, b, bin[j]) ;
                for (i = 0 ; i < 40 * bin[j] / binmax ; i++)
                        fputc ('*',fp) ;
                p_return (1) ;
                }
        pclose () ;
}

fom (number, loc, in, ysize, xsize)
        int                     ysize, xsize, number ;
        int                     loc[64][2] ;
        double                  in[64][16] ;
{
        double                  d1, d2, alpha, sum, var ;
        int                     i, j, k ;
        double                  max1, max2, nei, fig ;

        max1 = 10.0 * sqrt ((double) number_inputs) ;
        max2 = sqrt( pow( (double)ysize, 2.0) + pow( (double)xsize, 2.0) ) ;

        sum = 0.0 ;
        for (i = 0 ; i < number - 1 ; i++) {
                for (j = i+1 ; j < number ; j++) {
                        d1 = 0.0 ;
                        for (k = 0 ; k < number_inputs ; k++)
                                d1 += pow(in[i][k] - in[j][k] , 2.0) ;
                        d1 = sqrt ( d1 ) / max1 ;
                        d2 = sqrt ( pow((double) (loc[i][0] - loc[j][0]), 2.0) +
                        pow((double)(loc[i][1] - loc[j][1]), 2.0) ) / max2;
                        sum += pow (d1 - d2 , 2.0) ;
                        }
                }
        var = sqrt (sum) ;
        fig = var * ( 1.0 + nei) ;
        popen () ;
        p_return (2) ;
        fprintf (fp,"\nThe template test FOM is %7.4f", var) ;
        fprintf (fp,"\nThe neighborhood FOM is %7.4f", nei) ;
        fprintf (fp,"\nThe COMPOSITE FOM is %7.4f", fig) ;
        pfeed () ;
        pclose () ;
}

test_maxmin (dist, max, min)
        float                   dist ;
        float                   *max ;
        float                   *min ;
{
        if (dist < *min)
                *min = dist ;
        if (dist > *max)
                *max = dist ;
}

/*
********************************** mat2.c **************************************

        Routines to draw net diagrams (spectra), net trajectories, and
        graphics for net training routines. All graphics are performed
        using GKS routines.
*/

# include stdio
# include math
# include <gksdefs.h>
# include <descrip.h>

# define BLACK  0
# define WHITE  1

        float   ptsx[20][20][5], ptsy[20][20][5] ;
        float   px[20][20], py[20][20] ;
        int     used[20][20] ;
        int     colmat[20][20] ;
        int     pattern[16]=
                        {0, -15,-15, -12,-12, -11,-11, -1,-1, -2,-2, -4,-4, -5,-5, 1} ;

prepcolmat (ysize, xsize)
        int     xsize ;
        int     ysize ;
{
        int         r, c, xstart, ystart ;
        int         dx, dy ;

        dx = floor (550.0 / xsize) ;
        dy = floor (276.0 / ysize) ;

        xstart = 280 - dx * xsize / 2 ;
        ystart = 148 + dy * ysize / 2 ;

        for (c = 0 ; c < xsize ; c++) {
                for (r = 0 ; r < ysize ; r++) {
                        ptsx[r][c][4] = (ptsx[r][c][3] = (ptsx[r][c][0] =
                                xstart + c * dx)) ;
                        ptsx[r][c][2] = (ptsx[r][c][1] = ptsx[r][c][0] +dx- 1) ;
                        ptsy[r][c][4] = (ptsy[r][c][1] = (ptsy[r][c][0] =
                                ystart - r * dy)) ;
                        ptsy[r][c][3] = (ptsy[r][c][2] = ptsy[r][c][0] -dy+ 1) ;
                        px[r][c] = ptsx[r][c][0] + 8.0 ;
                        py[r][c] = ptsy[r][c][0] - 8.0 ;
                        }
                }
        for (r = 0 ; r < 20 ; r++) {
                for (c = 0 ; c < 20 ; c++) {
                        used[r][c] = 0 ;
                        }
                }
}

showem ()
{
        int         i ;
        float       rectx[16][6], recty[16][6] ;
        int         points = 5 ;

        for (i = 1 ; i < 16 ; i++) {
                rectx[i][4] = (rectx[i][3] = (rectx[i][0] = 173 + 16 * i)) ;
                rectx[i][2] = (rectx[i][1] = (rectx[i][0] + 15)) ;

                recty[i][4] = (recty[i][1] = (recty[i][0] = 30)) ;
                recty[i][3] = (recty[i][2] = 23) ;

                setfillstyle (pattern[i], i) ;
                gks$fill_area (&points, &rectx[i][0], &recty[i][0]) ;
                }
}

setfillstyle (pattern, pointer)
        int         pattern ;
        int         pointer ;
{
        int         style = 3 ;          /* hatch */
        int         color[16] = {0, 7,7, 3,3, 5,5, 4,4, 4, 6,6, 2,2, 1} ;

        if (pattern >= 0) {
                gks$set_fill_int_style (&pattern) ;
                }
        else {
                gks$set_fill_style_index (&pattern) ;
                gks$set_fill_int_style (&style) ;
                }
```

```
}

draw_net (number, loc)
        int         number, loc[64][2] ;
{
        int         i, old_value ;
        int         white = 0 ;
        int         black = 1 ;
        float       x, y ;
        char        s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d",-old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                sprintf (s,"%3d",i+1) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i+1 ;
                }
}

draw_neighbors (number, loc)
        int         number, loc[64][2] ;
{
        int         i, old_value ;
        int         white = 0 ;
        int         black = 1 ;
        float       x, y ;
        char        s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < 1 ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                sprintf (s,"%3d", (number+1)) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((number+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = number+1 ;
                }
}

draw_speech_map (number, loc)
        int     number, loc[125][2] ;
{
        int     i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y, xx[2], yy[2] ;
        int     points = 2 ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                gks$set_text_color_index (&black) ;
                sprintf (s,"%3d", i+1) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i+1 ;

                if (i != 0) {
                        xx[0] = px[loc[i-1][1]][loc[i-1][0]] ;
                        xx[1] = px[loc[i][1]][loc[i][0]] ;
                        yy[0] = py[loc[i-1][1]][loc[i-1][0]] ;
                        yy[1] = py[loc[i][1]][loc[i][0]] ;
                        gks$polyline (&points, xx, yy) ;
                        }
                }
}

draw_grid (ysize, xsize)
        int     xsize, ysize ;
{
        int     points = 5 ;
        int     r, c ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        gks$polyline (&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
}
```

```
draw_spectra (map, ysize, xsize)
        int     xsize, ysize ;
        float   map[20][20][16] ;
{
        int     points = 2 ;
        float   x[2], y[2] ;
        int     r, c, i ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        y[0] = ptsy[r][c][3] ;
                        for (i = 0 ; i < 15 ; i++) {
                                x[0] = ptsx[r][c][3] + 2.0 + (2.0 * i) ;
                                x[1] = x[0] ;
                                y[1] = y[0] + 16.0 * map[r][c][i] ;
                                gks$polyline (&points, x, y) ;
                                x[0] += 1.0 ;
                                x[1] += 1.0 ;
                                gks$polyline (&points, x, y) ;
                                }
                        }
                }
}

draw_grid2 (ysize, xsize, sub_title, length)
        int     xsize, ysize ;
        char    sub_title[30] ;
        short   length ;
{
        int     points = 5 ;
        int     r, c ;
        float   xloc, yloc ;
        struct dsc$descriptor title_dsc = { length,
                                        DSC$K_DTYPE_T,
                                        DSC$K_CLASS_S,
                                        sub_title } ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        gks$polyline (&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
        xloc = 277.0 - 3.0 * length ;
        yloc = 2.0 ;
        gks$text (&xloc, &yloc, &title_dsc) ;
}

statusem (gain, nrangey, nrangex, its)
        double  gain ;
        int     nrangey, nrangex ;
        long    its ;
{
        float   xloc, yloc ;
        char    s[60] ;
        $DESCRIPTOR(s_dsc,s) ;

        sprintf (s,
        "Gain = %4.2f Neighbors = %2d,%2d Iteration # %5ld",
        gain, nrangey, nrangex, its) ;
        xloc = 76.0 ;
        yloc = 2.0 ;
        gks$text (&xloc, &yloc, &s_dsc) ;
}

colorem (ysize, xsize)
        int       xsize, ysize ;
{
        int       r, c, color ;
        int       points = 5 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        color = colmat[r][c] ;
                        setfillstyle (pattern[color], color) ;
                        gks$fill_area(&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
}

pickcolors ()
{
        return ;
}
```

```
$ link neural4,nplot,nprinter,mat2,nweight4,options_file/opt
/*
********************************** neural4.c **********************************

        These routines will optionally recreate *.and files for replay, graph
        trajectories through nets, and create a record of the full trjectory
        reduction process in a *.trj file.
        **********************************************

                                Capt Gary Barmore, 7 Feb 88

*/

# include math
# include stdio
# include curses
# include time
# include <gksdefs.h>

# define bool       int

        float       map[20][20][16] ; /* output nodes */
        double      input[16] ; /* input nodes */

        int         xsize, ysize ; /* Size of array */
        int         number_inputs ;
        char        training_file[30] ;

mindist (map, inp, close)
        double              inp[16] ;
        int                 close[2] ;
        float               map[20][20][16] ;
{
        int                 r, c, i ;
        double              dot_product ;
        double              maximum = 0.0 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dot_product = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dot_product += inp[i] * map[r][c][i] ;
                        if (dot_product > maximum) {
                                maximum = dot_product ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}


main ()
{
        int         c ;

        printf ("\nNEURAL4 (Sound TRAJECTORIES!) ...\n") ;
        map_speech () ;
}
```

```
map_speech ()
{
        int             flag, r, c, i, j, k ;
        char            name[30], name2[30], sub_title[80], s[10], temp[30] ;
        char            name_trj[20] ;
        double          in[16], d1, d2, d3, d4, d5 ;
        float           element ;
        int             loc[125][2], loc2[125][2], loc3[125][2] ;
        FILE            *fsnd, *fnet, *fo ;
        int             sound, point, replica, x[5], y[5] ;
        short           length ;
        int             max_pts ;
        char            answer[10] ;
        int             snd_flag, fft_flag, graph_flag ;

        printf ("Do you want (0) sound file created or (1) not? ") ;
        scanf ("%d", &snd_flag) ;
        if (snd_flag = = 0){
                printf ("Enter name of training file used [less .trn]: ") ;
                scanf ("%s", temp) ;
                sprintf (training_file, "%s.trn", temp) ;

                printf ("Created with (0) FFT or (1) FFT2: ") ;
                scanf ("%d", &fft_flag) ;

                printf ("\nReading training file into memory!\n") ;
                read_trn_file () ;
                }

        printf ("Do you want (0) NO graphics or (1) TRAJECTORIES: ") ;
        scanf ("%d", &graph_flag) ;

        printf ("\nEnter name of net-file to use [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (name, "%s.net", temp) ;
        fnet = fopen (name, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r+ +) {
                for (c = 0 ; c < xsize ; c+ +) {
                        for (i = 0 ; i < number_inputs ; i+ +) {
                                fscanf (fnet," %f", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        if (snd_flag = = 0)
                correlate_sounds (map) ;

        for (;;) {
                printf ("\nEnter name [next] of speech file to map [less .trn]: ") ;
                scanf ("%s", temp) ;
                sprintf (name2, "%s.trn", temp) ;
                fsnd = fopen (name2, "r") ;
                sprintf (sub_title, "%s --> %s", name2, name) ;
                sprintf (name_trj, "%s.trj", temp) ;
                fo = fopen (name_trj, "w") ;

                if (graph_flag = = 1) {
                        graph_test (name) ;
                length = (short) strlen (sub_title) ;
                draw_grid2 (ysize, xsize, sub_title, length) ;
                }
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0) {
                        flag = 1 ;
                        }
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        mindist (map, in, &loc[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        if (graph_flag == 1) {
                draw_speech_map (sound, loc) ;
                scanf ("%s",s) ;
                clipoff () ;
                graphoff () ;
                }
        fprintf (fo, "%s\n", sub_title) ;
        fprintf (fo, "\nTrajectory through map: (%d)\n\n", sound) ;
        for (i = 0 ; i < sound ; i++) {
                point = loc[i][0] + loc[i][1] * xsize ;
                if ((i % 15) == 0)
                        fprintf (fo, "\n%3d ", point) ;
                else
                        fprintf (fo, "%3d ", point) ;
                }
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc[i][0] ;
                loc3[i][1] = loc[i][1] ;
                }
        max_pts = sound ;
        j = 0 ;
        if (j != 0)
                max_pts = j ;
        j = 0 ;
        for (k = 0 ; k < 3 ; k++) {
                x[k] = loc3[k][0] ;
                y[k] = loc3[k][1] ;
                }
        d1 = pow((double)(x[0]-x[2]), 2.0) +
                pow((double)(y[0]-y[2]), 2.0) ;
        d2 = pow((double)(x[0]-x[1]), 2.0) +
                pow((double)(y[0]-y[1]), 2.0) ;
        if ((d1 < 4.1) || (d2 < 4.1)) {
                loc2[j][0] = x[0] ;
                loc2[j][1] = y[0] ;
                j++ ;
                }
        for (i = 1 ; i < max_pts-1 ; i++) {
          if (i == 1){
                    for (k = 1 ; k < 5 ; k++) {
                              x[k] = loc3[i+k-2][0] ;
                              y[k] = loc3[i+k-2][1] ;
                              }
                    d2 = pow((double)(x[1]-x[2]), 2.0) +
                              pow((double)(y[1]-y[2]), 2.0) ;
                    d3 = pow((double)(x[3]-x[2]), 2.0) +
                              pow((double)(y[3]-y[2]), 2.0) ;
                    d4 = 5.0 ;
                    d5 = pow((double)(x[4]-x[2]), 2.0) +
                              pow((double)(y[4]-y[2]), 2.0) ;
                    }
          else if (i == max_pts-2) {
                    for (k = 0 ; k < 4 ; k++) {
                              x[k] = loc3[i+k-2][0] ;
                              y[k] = loc3[i+k-2][1] ;
                              }
                    d2 = pow((double)(x[1]-x[2]), 2.0) +
                              pow((double)(y[1]-y[2]), 2.0) ;
                    d3 = pow((double)(x[3]-x[2]), 2.0) +
                              pow((double)(y[3]-y[2]), 2.0) ;
                    d4 = pow((double)(x[0]-x[2]), 2.0) +
                              pow((double)(y[0]-y[2]), 2.0) ;
                    d5 = 5.0 ;
                    }
          else {
                    for (k = 0 ; k < 5 ; k++) {
                              x[k] = loc3[i+k-2][0] ;
                              y[k] = loc3[i+k-2][1] ;
                              }
                    d2 = pow((double)(x[2]-x[1]), 2.0) +
                              pow((double)(y[2]-y[1]), 2.0) ;
                    d3 = pow((double)(x[3]-x[2]), 2.0) +
                              pow((double)(y[3]-y[2]), 2.0) ;
                    d4 = pow((double)(x[2]-x[0]), 2.0) +
                              pow((double)(y[2]-y[0]), 2.0) ;
                    d5 = pow((double)(x[4]-x[2]), 2.0) +
                              pow((double)(y[4]-y[2]), 2.0) ;
                    }
          if ((d2 < 4.1) || (d3 < 4.1) || (d4 < 4.1)
                    || (d5 < 4.1)) {
                    loc2[j][0] = x[2] ;
                    loc2[j][1] = y[2] ;
                    j++ ;
                    }
          }
for (k = 0 ; k < 3 ; k++) {
          x[k] = loc3[max_pts+k-3][0] ;
          y[k] = loc3[max_pts+k-3][1] ;
          }
d1 = pow((double)(x[0]-x[2]), 2.0) +
          pow((double)(y[0]-y[2]), 2.0) ;
d3 = pow((double)(x[1]-x[2]), 2.0) +
          pow((double)(y[1]-y[2]), 2.0) ;
if ((d1 < 4.1) || (d3 < 4.1)) {
          loc2[j][0] = x[2] ;
          loc2[j][1] = y[2] ;
          j++ ;
          }
```

```
fprintf (fo, "\n\nAfter elimination of transients (%d) :\n\n", j) ;
for (i = 0 ; i < j ; i++) {
        point = loc2[i][0] + loc2[i][1] * xsize ;
        if ((i % 15) == 0)
                fprintf (fo, "\n%3d ", point) ;
        else
                fprintf (fo, "%3d ", point) ;
        }
for (i = 0 ; i < j ; i++) {
        loc3[i][0] = loc2[i][0] ;
        loc3[i][1] = loc2[i][1] ;
        }
max_pts = j ;
j = 0 ;
for (i = 0 ; i < max_pts ; i++) {
        if (i == 0) {
                for (k = 2 ; k < 5 ; k++) {
                        x[k] = loc3[k-2][0] ;
                        y[k] = loc3[k-2][1] ;
                        }
                d1 = 5.0 ; d2 = 5.0 ;
                d3 = pow((double)(x[3]-x[2]), 2.0) +
                        pow((double)(y[3]-y[2]), 2.0) ;
                d4 = pow((double)(x[4]-x[2]), 2.0) +
                        pow((double)(y[4]-y[2]), 2.0) ;
                }
        else if (i == max_pts-1) {
                for (k = 0 ; k < 3 ; k++) {
                        x[k] = loc3[i+k-2][0] ;
                        y[k] = loc3[i+k-2][1] ;
                        }
                d1 = pow((double)(x[0]-x[2]), 2.0) +
                        pow((double)(y[0]-y[2]), 2.0) ;
                d2 = pow((double)(x[1]-x[2]), 2.0) +
                        pow((double)(y[1]-y[2]), 2.0) ;
                d3 = 5.0 ; d4 = 5.0 ;
                }
        else if (i == 1) {
                for (k = 1 ; k < 5 ; k++) {
                        x[k] = loc3[i+k-2][0] ;
                        y[k] = loc3[i+k-2][1] ;
                        }
                d1 = pow((double)(x[1]-x[2]), 2.0) +
                        pow((double)(y[1]-y[2]), 2.0) ;
                d2 = 5.0 ;
                d3 = pow((double)(x[3]-x[2]), 2.0) +
                        pow((double)(y[3]-y[2]), 2.0) ;
                d4 = pow((double)(x[4]-x[2]), 2.0) +
                        pow((double)(y[4]-y[2]), 2.0) ;
                }
        else if (i == max_pts-2) {
                for (k = 0 ; k < 4 ; k++) {
                        x[k] = loc3[i+k-2][0] ;
                        y[k] = loc3[i+k-2][1] ;
                        }
                d1 = pow((double)(x[1]-x[2]), 2.0) +
                        pow((double)(y[1]-y[2]), 2.0) ;
                d2 = pow((double)(x[0]-x[2]), 2.0) +
                        pow((double)(y[0]-y[2]), 2.0) ;
                d4 = 5.0 ;
```

```
                        d3 = pow((double)(x[3]-x[2]), 2.0) +
                                pow((double)(y[3]-y[2]), 2.0) ;
                        }
                else {
                        for (k = 0 ; k < 5 ; k++) {
                                x[k] = loc3[i+k-2][0] ;
                                y[k] = loc3[i+k-2][1] ;
                                }
                        d1 = pow((double)(x[1]-x[2]), 2.0) +
                                pow((double)(y[1]-y[2]), 2.0) ;
                        d2 = pow((double)(x[0]-x[2]), 2.0) +
                                pow((double)(y[0]-y[2]), 2.0) ;
                        d3 = pow((double)(x[3]-x[2]), 2.0) +
                                pow((double)(y[3]-y[2]), 2.0) ;
                        d4 = pow((double)(x[4]-x[2]), 2.0) +
                                pow((double)(y[4]-y[2]), 2.0) ;
                        }
                if (((d1 < 4.1) && (d3 < 4.1)) ||
                        ((d1 < 4.1) && (d2 < 4.1)) ||
                        ((d3 < 4.1) && (d4 < 4.1))) {
                        loc2[j][0] = x[2] ;
                        loc2[j][1] = y[2] ;
                        j++ ;
                        }
                }
        fprintf (fo,"\n\n[Reduced Trajectory] Only three in a row! (%d)\n", j) ;
        for (i = 0 ; i < j ; i++) {
                point = loc2[i][0] + loc2[i][1] * xsize ;
                if ((i % 15) == 0)
                        fprintf (fo, "\n%3d ", point) ;
                else
                        fprintf (fo, "%3d ", point) ;
                }
        for (i = 0 ; i < j ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = j ;

        if (snd_flag == 0)
                create_snd_file (temp, loc2, max_pts, fft_flag) ;

        j = 0 ;
        loc2[j][0] = loc3[j][0] ;
        loc2[j][1] = loc3[j][1] ;
        for (i = 1 ; i < max_pts ; i++) {
                if ((loc3[i][0] != loc2[j][0]) ||
                        (loc3[i][1] != loc2[j][1])) {
                        j++ ;
                        loc2[j][0] = loc3[i][0] ;
                        loc2[j][1] = loc3[i][1] ;
                        }
                }
        j++ ;
        fprintf (fo, "\n\nNow listing final trajectory! (%d)\n", j) ;
        for (i = 0 ; i < j ; i++) {
                point = loc2[i][0] + loc2[i][1] * xsize ;
                if ((i % 15) == 0)
                        fprintf (fo, "\n%3d ", point) ;
                else
```

```
                                fprintf (fo, "%3d ", point) ;
                        }
                for (i = 0 ; i < j ; i++) {
                        loc3[i][0] = loc2[i][0] ;
                        loc3[i][1] = loc2[i][1] ;
                        }

                if (graph_flag == 1) {
                        max_pts = j ;
                        sprintf (sub_title,
                                "Reduced Trajectory: %s --> %s", name2, name) ;
                        graph_test (name) ;
                        length = (short) strlen (sub_title) ;
                        draw_grid2 (ysize, xsize, sub_title, length) ;
                        draw_speech_map (max_pts, loc2) ;
                        scanf ("%s",s) ;

                        clipoff () ;
                        graphoff () ;
                        }
                fclose (fo) ;
                }
}


graph_test (name)
        char    name[30] ;
{
        char    title[79], labelx[79] ;
        float   xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
        float   yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
        int     points = 5 ;
        int     ws_id = 1 ;
        int     clear_flag = 1 ;
        short   length ;

        sprintf(title,"NEURAL4: Kohonen %d x %d Neural Net -- %s",
                ysize, xsize, name) ;
        sprintf(labelx,"Noise Level : %g", noise) ;
        graphon () ;
        gks$clear_ws (&ws_id, &clear_flag) ;
        gks$polyline (&points, xloc, yloc) ;
        prepcolmat (ysize, xsize) ;
        length = (short) strlen (title) ;
        outtitle (title, length) ;
        length = (short) strlen (labelx) ;
        outlabelx (labelx, length) ;
        clipon () ;
}
```

```
$ link autodtw,options_file/opt
/*
****************************** autodtw.c ************************************

          Routines which use a first Kohonen neural net, files generated by
          autofft.exe (i.e. *.trn), header files (*.hdr) describing the
          contents of the *.trn files, and a header file describing the
          templates to perform Ney's one pass dynamic time warping algorithm.

          The algorithm allows recognition of both isolated and connected
          speech without changing the program. Additionally, the stretch
          factors are user selectable. 0.75 and 0.75 are suggested.

          A grader routine automatically scores the recognition process
          from the data given in the *.hdr files. Note that some bugs
          may still exit for grading isolated digits. Also note that
          the grader routine gives +1 for a correct digit, -1 for a wrong
          digit, -1 for a deleted digit, and -.5 for an additional digit.
          Recognized periods of silence are ignored.
          ****************************************************************
*/

# include stdio
# include math

          float     map[20][20][16] ; /* output nodes */
          int       xsize, ysize ; /* Size of array */
          int       number_inputs ;
          float     aa, bb ;
          char      training_file[30] ;
          int       num_templates ;
          float     total_digits ;
          float     wrong_digits ;
          float     min_dist ;
          int       tem_array[50], utt_array[50] ;
          int       t_words ;
          float     percent_corr, cum_per_corr ;

main ()
{
          char      temp[30], temp_file[30], utt_file[30] ;
          char      file_name[30], file_descr[80] ;
          char      sub[15][30] ;
          FILE      *flog, *fnet, *fstd, *fin, *ftmp ;
          char      template[15][30] ;
          int       t_array[15][200][2], u_array[2000][2], t_length[15] ;
          int       u_length, dist, r, c, i ;
          char      *std = "standard.hdr" ;
          int       entries_std, entries_cat ;
          int       i_std, i_cat ;


          printf ("\n\nAUTODTW: Tests standard set of utterances...\n\n") ;

          printf ("Enter name of template file [less .hdr]: ") ;
          scanf ("%s", temp) ;
          sprintf (temp_file, "%s.hdr", temp) ;
          ftmp = fopen (temp_file, "r") ;
          fscanf (ftmp, "%d", &num_templates) ;
          for (i = 0 ; i < num_templates ; i++)
```

```
            fscanf (ftmp, "%s", template[i]) ;
    fclose (ftmp) ;

    printf ("Enter name of log file [add .log]: ") ;
    scanf ("%s", temp) ;
    flog = fopen (temp, "w") ;
    fprintf (flog, "Log File: %s for AUTODTW.\n\n", temp) ;
    printf ("\nEnter horizontal weight: ") ;
    scanf ("%f", &aa) ;
    printf ("Enter vertical weight: ") ;
    scanf ("%f", &bb) ;
    fprintf (flog, "Horizontal and vertical weights are %g and %g.\n",
            aa, bb) ;
    printf("\nEnter name of net to use [less .net]: ") ;
    scanf ("%s", temp) ;
    sprintf (training_file, "%s.net", temp) ;
    fnet = fopen (trainin_file, "r") ;
    fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
    for (r = 0 ; r < ysize ; r++) {
            for (c = 0 ; c < xsize ; c++) {
                    for (i = 0 ; i < number_inputs ; i++) {
                            fscanf (fnet," %f", &map[r][c][i]) ;
                            }
                    }
            }
    fclose (fnet) ;
    for (i = 0 ; i < num_templates ; i++) {
            get_vectors(template[i], &t_array[i][0][0],
                    &t_length[i]) ;
            printf (" %s is %d vectors long\n",template[i],
                    t_length[i]) ;
            }
    fprintf(flog,"\nDynamic Time Warping using %s\n",training_file);

    fstd = fopen (std, "r") ;
    fscanf (fstd, "%d", &entries_std) ;
    for (i_std = 0 ; i_std < entries_std ; i_std++) {
            fscanf (fstd, "%s %s", file_name, file_descr) ;
            printf ("\n%s\n", file_descr) ;
            total_digits = 0.0 ;
            wrong_digits = 0.0 ;
            fprintf (flog, "\n%s\n", file_descr) ;
            fin = fopen (file_name, "r") ;
            fscanf (fin, "%d", &entries_cat) ;
            for (i_cat = 0 ; i_cat < entries_cat ; i_cat++) {
                    fscanf (fin, "%s %d", utt_file, &t_words) ;
                    for (i = 0 ; i < t_words ; i++)
                            fscanf (fin, "%d", &tem_array[i]) ;
                    get_vectors(utt_file, u_array, &u_length) ;
                    printf (" %-12s is: ",
                            utt_file) ;
                    fprintf (flog," %-12s is: ",
                            utt_file) ;
                    cdtw (t_array,u_array,t_length,u_length,flog) ;
                    }
            fclose (fin) ;
            }
    fclose (fstd) ;
    fclose (flog) ;
}
```

```
cdtw (t_array, u_array, t_length, u_length, flog)
        int     t_array[15][200][2], u_array[2000][2] ;
        int     t_length[15], u_length ;
        FILE    *flog ;
{
        float   accum_dist[2][15][200] ;
        int     utterance[50] ;
        int     ptr, b_ptr ;
        int     back_ptr[2][15][200] ;
        int     from_template[2000] ;
        int     from_frame[2000] ;
        int     i, j, k, kk ;
        float   d1, d2, d3, dist ;
        float   min_dist ;

        ptr = 0 ;
        for (k = 0 ; k < num_templates ; k++) {
                for (j = 0 ; j < t_length[k] ; j++) {
                        if (j == 0) {
                                accum_dist[ptr][k][j] = (dist =
                                        abs(u_array[0][0]- t_array[k][j][0]) +
                                        abs(u_array[0][1]- t_array[k][j][1]));
                                }
                        else {
                                accum_dist[ptr][k][j] = (dist += bb * (
                                        abs(u_array[0][0] - t_array[k][j][0]) +
                                        abs(u_array[0][1] - t_array[k][j][1])));
                                }
                        back_ptr[ptr][k][j] = 0 ;
                        }
                }
        for (i = 1 ; i < u_length ; i++) {
                if (ptr == 0) {
                        ptr = 1 ;
                        b_ptr = 0 ;
                        }
                else {
                        ptr = 0 ;
                        b_ptr = 1 ;
                        }
                for (k = 0 ; k < num_templates ; k++) {
                for (j = 0 ; j < t_length[k] ; j++) {
                        dist = abs(u_array[i][0] - t_array[k][j][0]) +
                                abs(u_array[i][1] - t_array[k][j][1]) ;
                        if (j == 0) {
                                min_dist = 99999.0 ;
                                for (kk = 0 ; kk < num_templates ; kk++) {
                                if (min_dist >
                                accum_dist[b_ptr][kk][t_length[kk]-1])
                                        min_dist =
                                        accum_dist[b_ptr][kk][t_length[kk]-1];
                                        }
                                if (accum_dist[b_ptr][k][0] < min_dist) {
                                        accum_dist[ptr][k][0] = aa * dist +
                                        accum_dist[b_ptr][k][0] ;
                                        back_ptr[ptr][k][0] =
                                        back_ptr[b_ptr][k][0] ;
                                        }
                                else {
```

```
                                accum_dist[ptr][k][0] =
                                min_dist + dist ;
                                back_ptr[ptr][k][0] = i-1 ;
                                }
                        }
                else {
                        d1 = accum_dist[b_ptr][k][j-1] + dist ;
                        d2 = accum_dist[ptr][k][j-1] + (bb * dist) ;
                        d3 = accum_dist[b_ptr][k][j] + (aa * dist) ;
                        if (d2 <= d3 && d2 < d1) {
                                accum_dist[ptr][k][j] = d2 ;
                                back_ptr[ptr][k][j] =
                                        back_ptr[ptr][k][j-1] ;
                                }
                        else if (d3 <= d2 && d3 < d1) {
                                accum_dist[ptr][k][j] = d3 ;
                                back_ptr[ptr][k][j] =
                                        back_ptr[b_ptr][k][j] ;
                                }
                        else {
                                accum_dist[ptr][k][j] = d1 ;
                                back_ptr[ptr][k][j] =
                                        back_ptr[b_ptr][k][j-1] ;
                                }
                        }
                }
        }
        min_dist = 99999.0 ;
        for (k = 0 ; k < num_templates ; k++) {
                if (min_dist > accum_dist[ptr][k][t_length[k]-1]) {
                        min_dist = accum_dist[ptr][k][t_length[k]-1] ;
                        kk = k ;
                        }
                }
        from_template[i] = kk ;
        from_frame[i] = back_ptr[ptr][kk][t_length[kk]-1] ;
        }
ptr = u_length - 1 ;
i = -1 ;
while ((ptr > 0) && (i < 49)) {
        utterance[++i] = from_template[ptr] ;
        ptr = from_frame[ptr] ;
        }
k = 0 ;
for (j = i ; j >= 0 ; j--) {
        if (utterance[j] == 10) {
                printf (".. ") ;
                fprintf (flog, ".. ") ;
                }
        else {
                printf ("%d ", utterance[j]) ;
                fprintf (flog, "%d ", utterance[j]) ;
                utt_array[k++] = utterance[j] ;
                }
        }
printf ("\n Should be: ") ;
fprintf (flog, "\n Should be: ") ;
for (j = 0 ; j < t_words ; j++) {
        printf ("%d ", tem_array[j]) ;
        fprintf (flog, "%d ", tem_array[j]) ;
```

```
                        }
        grader (tem_array, utt_array, t_words, k) ;
        printf ("\n correct = %5.3f cum_correct = %5.3f\n",
                percent_corr, cum_per_corr) ;
        fprintf (flog, "\n correct = %5.3f cum_correct = %5.3f\n",
                percent_corr, cum_per_corr) ;
}

mindist (map, inp, close)
        double                  inp[16] ;
        int                     close[2] ;
        float                   map[20][20][16] ;
{
        int                     r, c, i ;
        double                  dot_product ;
        double                  maximum = 0.0 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dot_product = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                dot_product += inp[i] * map[r][c][i] ;
                        if (dot_product > maximum) {
                        maximum = dot_product ;
                        close[0] = c ;
                        close[1] = r ;
                        }
                }
        }
}

get_vectors (name, array, length)
        char    name[30] ;
        int     array[2000][2] ;
        int     *length ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;

        int     loc2[2000][2], loc3[2000][2] ;
        FILE    *fsnd ;
        int     sound, point, x[5], y[5] ;
        int     max_pts ;


        fsnd = fopen (name, "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0) {
                        flag = 1 ;
                        }
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        mindist (map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fand) ;
        for (i = 0 ; i < sound ; i++) {
                point = loc2[i][0] + loc2[i][1] * xsize ;
                }
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;
        j = 0 ;

        if (j != 0)
                max_pts = j ;
        j = 0 ;
        for (k = 0 ; k < 3 ; k++) {
                x[k] = loc3[k][0] ;
                y[k] = loc3[k][1] ;
                }
        d1 = pow((double)(x[0]-x[2], 2.0) +
                pow((double)(y[0]-y[2]), 2.0) ;
        d2 = pow((double)(x[0]-x[1]), 2.0) +
                pow((double)(y[0]-y[1]), 2.0) ;
        if ((d1 < 4.1) || (d2 < 4.1)) {
                loc2[j][0] = x[0] ;
                loc2[j][1] = y[0] ;
                j++ ;
                }
        for (i = 1 ; i < max_pts-1 ; i++) {
                if (i == 1){
                        for (k = 1 ; k < 5 ; k++) {
                                x[k] = loc3[i+k-2][0] ;
                                y[k] = loc3[i+k-2][1] ;
                                }
                        d2 = pow((double)(x[1]-x[2]), 2.0) +
                                pow((double)(y[1]-y[2]), 2.0) ;
                        d3 = pow((double)(x[3]-x[2]), 2.0) +
                                pow((double)(y[3]-y[2]), 2.0) ;
                        d4 = 5.0 ;
                        d5 = pow((double)(x[4]-x[2]), 2.0) +
                                pow((double)(y[4]-y[2]), 2.0) ;
                        }
                else if (i == max_pts-2) {
                        for (k = 0 ; k < 4 ; k++) {
                                x[k] = loc3[i+k-2][0] ;
                                y[k] = loc3[i+k-2][1] ;
                                }
                        d2 = pow((double)(x[1]-x[2]), 2.0) +
                                pow((double)(y[1]-y[2]), 2.0) ;
                        d3 = pow((double)(x[3]-x[2]), 2.0) +
                                pow((double)(y[3]-y[2]), 2.0) ;
                        d4 = pow((double)(x[0]-x[2]), 2.0) +
                                pow((double)(y[0]-y[2]), 2.0) ;
                        d5 = 5.0 ;
                        }
                else {
```

```
        for (k = 0 ; k < 5 ; k++) {
            x[k] = loc3[i+k-2][0] ;
            y[k] = loc3[i+k-2][1] ;
            }
        d2 = pow((double)(x[2]-x[1]), 2.0) +
            pow((double)(y[2]-y[1]), 2.0) ;
        d3 = pow((double)(x[3]-x[2]), 2.0) +
            pow((double)(y[3]-y[2]), 2.0) ;
        d4 = pow((double)(x[2]-x[0]), 2.0) +
            pow((double)(y[2]-y[0]), 2.0) ;
        d5 = pow((double)(x[4]-x[2]), 2.0) +
            pow((double)(y[4]-y[2]), 2.0) ;
        }
    if ((d2 < 4.1) || (d3 < 4.1) || (d4 < 4.1)
        || (d5 < 4.1)) {
        loc2[j][0] = x[2] ;
        loc2[j][1] = y[2] ;
        j++ ;
        }
    }
for (k = 0 ; k < 3 ; k++) {
    x[k] = loc3[max_pts+k-3][0] ;
    y[k] = loc3[max_pts+k-3][1] ;
    }
d1 = pow((double)(x[0]-x[2]), 2.0) +
    pow((double)(y[0]-y[2]), 2.0) ;
d3 = pow((double)(x[1]-x[2]), 2.0) +
    pow((double)(y[1]-y[2]), 2.0) ;
if ((d1 < 4.1) || (d3 < 4.1)) {
    loc2[j][0] = x[2] ;
    loc2[j][1] = y[2] ;
    j++ ;
    }
for (i = 0 ; i < j ; i++) {
    point = loc2[i][0] + loc2[i][1] * xsize ;
    }
for (i = 0 ; i < j ; i++) {
    loc3[i][0] = loc2[i][0] ;
    loc3[i][1] = loc2[i][1] ;
    }

max_pts = j ;
j = 0 ;
for (i = 0 ; i < max_pts ; i++) {
    if (i == 0) {
        for (k = 2 ; k < 5 ; k++) {
            x[k] = loc3[k-2][0] ;
            y[k] = loc3[k-2][1] ;
            }
        d1 = 5.0 ; d2 = 5.0 ;
        d3 = pow((double)(x[3]-x[2]), 2.0) +
            pow((double)(y[3]-y[2]), 2.0) ;
        d4 = pow((double)(x[4]-x[2]), 2.0) +
            pow((double)(y[4]-y[2]), 2.0) ;
        }
    else if (i == max_pts-1) {
        for (k = 0 ; k < 3 ; k++) {
            x[k] = loc3[i+k-2][0] ;
            y[k] = loc3[i+k-2][1] ;
            }
            d1 = pow((double)(x[0]-x[2]), 2.0) +
                pow((double)(y[0]-y[2]), 2.0) ;
            d2 = pow((double)(x[1]-x[2]), 2.0) +
                pow((double)(y[1]-y[2]), 2.0) ;
            d3 = 5.0 ; d4 = 5.0 ;
            }
        else if (i = = 1) {
            for (k = 1 ; k < 5 ; k+ +) {
                x[k] = loc3[i+k-2][0] ;
                y[k] = loc3[i+k-2][1] ;
                }
            d1 = pow((double)(x[1]-x[2]), 2.0) +
                pow((double)(y[1]-y[2]), 2.0) ;
            d2 = 5.0 ;
            d3 = pow((double)(x[3]-x[2]), 2.0) +
                pow((double)(y[3]-y[2]), 2.0) ;
            d4 = pow((double)(x[4]-x[2]), 2.0) +
                pow((double)(y[4]-y[2]), 2.0) ;
            }
        else if (i = = max_pts-2) {
            for (k = 0 ; k < 4 ; k+ +) {
                x[k] = loc3[i+k-2][0] ;
                y[k] = loc3[i+k-2][1] ;
                }
            d1 = pow((double)(x[1]-x[2]), 2.0) +
                pow((double)(y[1]-y[2]), 2.0) ;
            d2 = pow((double)(x[0]-x[2]), 2.0) +
                pow((double)(y[0]-y[2]), 2.0) ;
            d4 = 5.0 ;
            d3 = pow((double)(x[3]-x[2]), 2.0) +
                pow((double)(y[3]-y[2]), 2.0) ;
            }
        else {
            for (k = 0 ; k < 5 ; k+ +) {
                x[k] = loc3[i+k-2][0] ;
                y[k] = loc3[i+k-2][1] ;
                }
            d1 = pow((double)(x[1]-x[2]), 2.0) +
                pow((double)(y[1]-y[2]), 2.0) ;
            d2 = pow((double)(x[0]-x[2]), 2.0) +
                pow((double)(y[0]-y[2]), 2.0) ;
            d3 = pow((double)(x[3]-x[2]), 2.0) +
                pow((double)(y[3]-y[2]), 2.0) ;
            d4 = pow((double)(x[4]-x[2]), 2.0) +
                pow((double)(y[4]-y[2]), 2.0) ;
            }
        if (((d1 < 4.1) && (d3 < 4.1)) | |
            ((d1 < 4.1) && (d2 < 4.1)) | |
            ((d3 < 4.1) && (d4 < 4.1))) {
            loc2[j][0] = x[2] ;
            loc2[j][1] = y[2] ;
            j+ + ;
            }
        }
    *length = j ;
    for (i = 0 ; i < j ; i+ +) {
        array[i][0] = loc2[i][0] ;
        array[i][1] = loc2[i][1] ;
        }
}
```

```
grader (t_array, u_array, t_length, u_length)
        int     t_array[50], u_array[50] ;
        int     t_length, u_length ;
{
        int     i, j, ptr, b_ptr ;
        float   back_ptr[2][50] ;
        float   d, min, dist, back ;

        back = 0.0 ;
        ptr = 1 ;
        for (i = 0 ; i < u_length ; i++) {
                if (ptr == 0){
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                else {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                if (i == 0) {
                        for (j = 0 ; j < t_length ; j++) {
                                if (t_array[j] == u_array[0])
                                        dist = 0.0 + back ;
                                else
                                        dist = 1.0 + back ;
                                back = dist ;
                                back_ptr[ptr][j] = dist ;
                                }
                        }
                else {
                        for (j = 0 ; j < t_length ; j++) {
                                if (t_array[j] == u_array[i])
                                        dist = 0.0 ;
                                else
                                        dist = 1.0 ;
                                if (j == 0)
                                        back_ptr[ptr][0] = back_ptr[b_ptr][0] +
                                                (0.5 * dist) ;
                                else {
                                        min = back_ptr[b_ptr][j] + (0.5*dist) ;
                                        d = back_ptr[b_ptr][j-1] + dist ;
                                        if (d < min)
                                                min = d ,
                                        d = back_ptr[ptr][j-1] + dist ;
                                        if (d < min)
                                                min = d ;
                                        back_ptr[ptr][j] = min ;
                                        }
                                }
                        }
                }
        if (u_length > 0)
                min_dist = back_ptr[ptr][t_length-1] ;
        else
                min_dist = t_length ;
        percent_corr = (t_length - min_dist) / t_length ;
        total_digits += t_length ;
        wrong_digits += min_dist ;
        cum_per_corr = (total_digits - wrong_digits) / total_digits ;
}
```

```
$ link twokoh4,nweight8,options_file/opt
/*
******************************** twokoh4.c ********************************

        Routines to train a second Kohonen net to process reduced
        trajectories from first kohonen net. Number of points in the
        trajectory is 75. Each point is represented as a scalar from 0
        to 224. Short trajectories are filled with trailing 0's.
        *************************************************************

                         Capt Gary Barmore, 8 Jul 88
*/

# include math
# include stdio
# include time

        float     map[20][20][225] ; /* output nodes */
        double    input[225] ; /* input nodes */
        double    gain, noise ;
        double    mcount ;
        double    percent ;

        int       closest[2] ; /* closest node */
        int       neigh[2] ; /* neighbor */
        int       nrangex, nrangey ; /* neighbor range */
        int       nfactorx, nfactory ; /* neighbor factor */
        long      count ; /* # of iterations */
        int       graph ; /* # between plots */
        int       seed ;
        int       maxneighx, maxneighy ; /* Starting area */
        int       minneighx, minneighy ; /* Final area */
        int       xsize, ysize ; /* Size of array */
        int       number_inputs ;

        char      training_file[30], net_file[30], first_net_file[30] ;
        char      temp_file[15] ;
        char      net_name[15] ;

        struct curve {
                  int             type ;
                  double          maxgain ;
                  double          mingain ;
                  double          midgain ;
                  int             midtime ;
                  } gcurve ;

        struct flg {
                  int             rnd_in ;
                  } flag ;

init (map)
        float     map[20][20][225] ;
{
        int       r, c, i ;
        float     max_rand = pow(2.0, 31.0) - 1.0 ;

        for (r = 0 ; r < ysize ; r++) {
                  for (c = 0 ; c < xsize ; c++) {
                            for (i = 0 ; i < number_inputs ; i++) {
                                map[r][c][i] = rand () / max_rand ;
                                }
                        }
                }
}

mindist (map, inp, close)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 999999.0 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}

userinp ()
{
        int             line ;
        int             c ;
        struct tm       *localtime(), *time ;
        int             *bintim ;

        do {
                printf ("TWOKOH4 net training (no graphics)... \n\n") ;

                printf("Enter size 'm n' (for an m x n) of array = ? [int int] ") ;
                scanf("%d %d", &ysize, &xsize) ;
                if (ysize < 2)
                        ysize = 2 ;
                else if (ysize > 20)
                        ysize = 20 ;
                if (xsize < 2)
                        xsize = 2 ;
                else if (xsize > 20)
                        xsize = 20 ;

                printf ("Do you want 0) sequential training,\n") ;
                printf (" 1) randomized training? ") ;
                scanf ("%d", &flag.rnd_in) ;
                printf ("Enter name of header file containing words (less .hdr): ") ;
                scanf ("%s", temp_file) ;
                sprintf (training_file, "%s.hdr", temp_file) ;

                number_inputs = 75 ;

                printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
        scanf ("%s", net_name) ;
        sprintf (first_net_file, "%s.net", net_name) ;

        printf("Enter name of net file to create [less .net]: ") ;
        scanf ("%s", net_name) ;
        sprintf (net_file, "%s.net", net_name) ;

        printf ("Number of iterations = ? [int] ") ;
        scanf ("%ld", &count) ;
        if (count < = 10 || count > 130000)
            count = 100 ;
        mcount = (double) count ;

        printf ("Number of iterations between status messages = ? [int] ") ;
        scanf ("%d", &graph) ;
        if (graph < 1 || graph > count)
            graph = 10 ;

        ingain () ;

        printf ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
        scanf ("%d %d", &maxneighy, &maxneighx) ;
        if (maxneighx < 2 || maxneighx > xsize - 1)
            maxneighx = 2 ;
        if (maxneighy < 2 || maxneighy > ysize - 1)
            maxneighy = 2 ;

        printf ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
        scanf ("%d %d", &minneighy, &minneighx) ;
        if (minneighx < 1 || minneighx > maxneighx)
            minneighx = 1 ;
        if (minneighy < 1 || minneighy > maxneighy)
            minneighy = 1 ;

        printf
            ("Initial seed for random # generator (0 SELECTS TIME) = ? [int] ");
        scanf ("%d", &seed) ;
        if (seed = = 0) {
            time = localtime (bintim) ;
            time.tm_sec %= 60 ;
            time.tm_min %= 60 ;
            seed = time.tm_sec * time.tm_min ;
            }
        srand (seed) ;

        printf("Ready to begin? (y/n) ") ;
        while ((c = getc (stdin)) = = ' ' || c = = '\n' || c = = '\t')
            ;
        } while (c != 'y') ;
}

ingain ()
{
    int                 line ;

    printf("For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR : ");
    scanf ("%d", &gcurve.type) ;

    if (gcurve.type = = 0 || gcurve.type = = 1) {
        printf ("Maximum gain = ? [float] ") ;
```

```
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ;

                printf ("Minimum gain = ? [float] ") ;
                scanf ("%E", &gcurve.mingain) ;
                if (gcurve.mingain <= 0.0 || gcurve.mingain >= 1.0)
                        gcurve.mingain = 0.0 ;
                }
        else {
                printf ("First segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ;

                printf ("Second segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.midgain) ;
                if (gcurve.midgain <= 0.0 || gcurve.midgain >= 1.0)
                        gcurve.midgain = 0.0 ;

                printf ("Second segment starting iteration = ? [float] ") ;
                scanf ("%d", &gcurve.midtime) ;
                if (gcurve.midtime <= 0 || gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long    i ;
{
        if (gcurve.type == 0)
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) +
        gcurve.mingain ;
        else if (gcurve.type == 1)
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i -
                        count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
        }
}

save_net ()
{
        int             r, c, i ;
        FILE            *fnet ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long                    i ;
        char                    s1[10] ;
        int                     ws_id = 1 ;
        int                     clear_flag = 1;
        FILE                    *tf ;
        extern unsigned         _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */
        read_trn_file () ;

        for (i = 1 ; i < = count ; i+ +) {
                if (i % graph = = 0) {
                        printf ("TWOKOH4: gain = %f, yrange = %d, ",
                                gain, nrangey) ;
                        printf ("xrange = %d, iteration # %d", nrangex,i) ;
                        printf (" (of %ld)\n", count) ;
                        if (access (net_file,0) = = 0)
                                delete (net_file) ;
                        save_net () ;
                        }
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in = = 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (map, input, closest) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                        ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
********************************* nweight8.c *********************************
```

```
        These routines allow training and testing of a second Kohonen
        net of two net system. Inputs are 75 point reduced trajectories
        from the first net. Each point is a scalar from 0 to 224. Short
        trajectories are filled with trailing 0's.
        ****************************************************************
*/

# include math
# include stdio
# include stat

        extern double           input[225] ; /* input nodes */
        extern double           gain ;
        extern int              closest[2] ; /* closest node */
        extern int              neigh[2] ; /* neighbor */
        extern int              xsize, ysize ; /* Size of array */
        extern int              number_inputs ;
        extern char             training_file[30] ;
        extern char             first_net_file[30] ;

        int                     number_discretes ;
        int                     word_counter ;
        int                     num_words ;
        char                    word_number[100][15] ;
        int                     f_ysize, f_xsize, f_number_inputs ;
        float                   f_map[20][20][16] ;

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     i, r, c ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;
        word_counter = 0 ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}
```

```
get_rnd_in ()
{
        int                     i ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

weightem (map)
        float                   map[20][20][225] ;
{
        int                     nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
                if (nright >= xsize)
                        nright = xsize - 1 ;
                nleft = closest[0] - neigh[0] + 1 ;
                if (nleft < 0)
                        nleft = 0 ;
                nup = closest[1] - neigh[1] + 1 ;
                if (nup < 0)
                        nup = 0 ;
                ndown = closest[1] + neigh[1] - 1 ;
                if (ndown >= ysize)
                        ndown = ysize - 1 ;
                }
        else {
                nright = closest[0] ;
                nleft = closest[0] ;
                nup = closest[1] ;
                ndown = closest[1] ;
                }

        for (r = nup; r <= ndown ; r++) {
                for (c = nleft ; c <= nright ; c++) {
                        for (i = 0 ; i < number_inputs ; i++)
                                map[r][c][i] += gain * (input[i] - map[r][c][i]) ;
                        }
                }
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;

        int     loc2[2000][2], loc3[2000][2] ;
        FILE    *fsnd ;
        int     sound, point, x[5], y[5] ;
        int     max_pts ;
        double  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                point = loc2[i][0] + loc2[i][1] * f_xsize ;
                input[i] = point / 225.0 ;
                }
}


f_mindist (f_map, inp, close)
        double                  inp[16] ;
        int                     close[2] ;
        float                   f_map[20][20][16] ;
{
        int                     r, c, i ;
        double                  dist ;
        double                  minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link twobas2,nweight10,options_file/opt
/*
******************************* twobas2.c ********************************

        Routine to train a second Kohonen net to process reduced 75
        point trajectories from first kohonen net. The points are scalars
        from 0 to 224. Short trajectories are filled wih trailing 0's.

        Initial weights are randomly distorted from the first run of
        inputs. Updates to weights use trajectories precalculated and
        stored in path.dat file.

        This version adds 'conscience' to chooseing closest node; i.e. if
        node is   chosen too often it is not considered for closest status.
        **************************************************************

                                Capt Gary Barmore, 10 Aug 88
*/

# include math
# include stdio
# include time

        int     conscience[20][20] ;/* records # times closest */
        int     nodes ;                          /* number of nodes */
        double  consc = 1.1 ;           /* conscience factor */
        float   map[20][20][225] ; /* output nodes */
        double  input[225] ; /* input nodes */
        double  gain, noise ;
        double  mcount ;
        double  percent ;

        int     closest[2] ; /* closest node */
        int     neigh[2] ; /* neighbor */
        int     nrangex, nrangey ; /* neighbor range */
        int     nfactorx, nfactory ; /* neighbor factor */
        lon     gcount ; /* # of iterations */
        int     graph ; /* # between plots */
        int     seed ;
        int     maxneighx, maxneighy ; /* Starting area */
        int     minneighx, minneighy ; /* Final area */
        int     xsize, ysize ; /* Size of array */
        int     number_inputs ;

        char    training_file[30], net_file[30], first_net_file[30] ;
        char    temp_file[15] ;
        char    net_name[15] ;

        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        struct flg {
                int             rnd_in ;
                } flag ;
```

```
init (map)
        float           map[20][20][225] ;
{
        int             r, c, i ;
        float           max_rand = pow(2.0, 31.0) - 1.0 ;

        nodes = ysize * xsize ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        conscience[r][c] = 0 ;
                        getin () ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                map[r][c][i] = input[i] +
                                        rand () / max_rand ;
                                }
                        }
                }
}

mindist (map, inp, close, its)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
        long            its ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        if (conscience[r][c] < consc * its / nodes ) {
                                for (i = 0 ; i < number_inputs ; i++)
                                        dist += pow(inp[i] - map[r][c][i], 2.0);
                                if (dist < minimum) {
                                        minimum = dist ;
                                        close[0] = c ;
                                        close[1] = r ;
                                        }
                                }
                        }
                }
        conscience[close[1]][close[0]] += 1 ;
}

userinp ()
{
        int             line ;
        int             c ;
        struct tm       *localtime(), *time ;
        int             *bintim ;

        do {
                printf ("TWO KOHonen net training (output only)... \n\n") ;

                printf("Enter size 'm n' (for an m x n) of array = ? [int int] ") ;
                scanf("%d %d", &ysize, &xsize) ;
                if (ysize < 2)
                        ysize = 2 ;
else if (ysize > 20)
        ysize = 20 ;
if (xsize < 2)
        xsize = 2 ;
else if (xsize > 20)
        xsize = 20 ;

printf ("Do you want 0) sequential training,\n") ;
printf (" 1) randomized training? ") ;
scanf ("%d", &flag.rnd_in) ;
printf ("Enter name of header file containing words (less .hdr): ") ;
scanf ("%s", temp_file) ;
sprintf (training_file, "%s.hdr", temp_file) ;

number_inputs = 75 ;

printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
scanf ("%s", net_name) ;
sprintf (first_net_file, "%s.net", net_name) ;

printf("Enter name of net file to create [less .net]: ") ;
scanf ("%s", net_name) ;
sprintf (net_file, "%s.net", net_name) ;

printf ("Number of iterations = ? [int] ") ;
scanf ("%ld", &count) ;
if (count <= 10 || count > 130000)
        count = 100 ;
mcount = (double) count ;

printf ("Number of iterations between status messages = ? [int] ") ;
scanf ("%d", &graph) ;
if (graph < 1 || graph > count)
        graph = 10 ;

ingain () ;

printf ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &maxneighy, &maxneighx) ;
if (maxneighx < 2 || maxneighx > xsize - 1)
        maxneighx = 2 ;
if (maxneighy < 2 || maxneighy > ysize - 1)
        maxneighy = 2 ;

printf ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &minneighy, &minneighx) ;
if (minneighx < 1 || minneighx > maxneighx)
        minneighx = 1 ;
if (minneighy < 1 || minneighy > maxneighy)
        minneighy = 1 ;

printf
        ("Initial seed for random # generator = ? [int] ");
scanf ("%d", &seed) ;
if (seed == 0) {
        seed = 138 ;
        }
srand (seed) ;

printf("Ready to begin? (y/n) ") ;
```

```
                while ((c = getc (stdin)) == ' ' || c == '\n' || c == '\t')
                        ;
                } while (c != 'y') ;
}

ingain ()
{
        int             line ;

        printf("For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR : ");
        scanf ("%d", &gcurve.type) ;

        if (gcurve.type == 0 || gcurve.type == 1) {
                printf ("Maximum gain = ? [float] ") ;
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ;
                printf ("Minimum gain = ? [float] ") ;
                scanf ("%E", &gcurve.mingain) ;
                if (gcurve.mingain <= 0.0 || gcurve.mingain >= 1.0)
                        gcurve.mingain = 0.0 ;
                }
        else {
                printf ("First segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ,

                printf ("Second segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.midgain) ;
                if (gcurve.midgain <= 0.0 || gcurve.midgain >= 1.0)
                        gcurve.midgain = 0.0 ;

                printf ("Second segment starting iteration = ? [float] ") ;
                scanf ("%d", &gcurve.midtime) ;
                if (gcurve.midtime <= 0 || gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long            i ;
{
        if (gcurve.type == 0)
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) +
        gcurve.mingain ;
        else if (gcurve.type == 1)
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i -
                        count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
                }
}

save_net ()
```

```
{
        int                     r, c, i ;
        FILE                    *fnet ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long                    i ;
        char                    s1[10] ;
        int                     ws_id = 1 ;
        int                     clear_flag = 1;
        FILE                    *tf ;
        extern unsigned         _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */
        read_trn_file () ;

        for (i = 1 ; i <= count ; i++) {
                if (i % graph == 0) {
                        printf ("TWOKOH7: gain = %f, yrange = %d, ",
                                gain, nrangey) ;
                        printf ("xrange = %d, iteration # %d", nrangex,i) ;
                        printf (" (of %ld)\n", count) ;
                        if (access (net_file,0) == 0)
                                delete (net_file) ;
                        save_net () ;
                        }
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in == 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (map, input, closest, i) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
******************************** nweight10.c ********************************

        These routines allow training and testing of a second Kohonen
        net of two net system. Inputs are 75 point trajectories from the
        first net. Points on the trajectories are scalar values from 0 to
        224. Short trajectories are filled with trailing 0's.

        This version uses a set of input trajectories saved in path.dat
        to save time reading *.trn files and running them through the
        first trajectory on every pass.
        ************************************************************
*/

# include math
# include stdio
# include stat

        double          innput[100][75] ;/* input vectors */

        extern double   input[225] ; /* input nodes */
        extern double   gain ;
        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;
        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

read_trn_file ()
{
        FILE            *tf, *fnet ;
        int             j, i, r, c ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        fnet = fopen ("path.dat", "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                for (j = 0 ; j < number_inputs ; j++)
                        fscanf (fnet, "%le", &innput[i][j]) ;
                }
        fclose (fnet) ;
        word_counter = 0 ;
}

getin ()
{
        int     j ;

        if (word_counter == num_words)
                word_counter = 0 ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
        word_counter++ ;
}

get_rnd_in ()
{
        int             i, j ;
        double          max_rand = pow (2.0, 31.0) - 1.0 ;
        int             pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
}

weightem (map)
        float           map[20][20][225] ;
{
        int             nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
        if (nright >= xsize)
                nright = xsize - 1 ;
        nleft = closest[0] - neigh[0] + 1 ;
        if (nleft < 0)
                nleft = 0 ;
        nup = closest[1] - neigh[1] + 1 ;
        if (nup < 0)
                nup = 0 ;
        ndown = closest[1] + neigh[1] - 1 ;
        if (ndown >= ysize)
                ndown = ysize - 1 ;
        }
```

```
            else {
                        nright = closest[0] ;
                        nleft = closest[0] ;
                        nup = closest[1] ;
                        ndown = closest[1] ;
                        }
            for (r = nup; r <= ndown ; r++) {
                        for (c = nleft ; c <= nright ; c++) {
                                    for (i = 0 ; i < number_inputs ; i++)
                                                map[r][c][i] += gain * (input[i] - map[r][c][i]) ;
                                    }
                        }
}

read_word (pointer)
            int         pointer ;
{
            int         flag, r, c, i, j, k ;
            double      in[16], d1, d2, d3, d4, d5 ;
            float       element ;

            int         loc2[2000][2], loc3[2000][2] ;
            FILE        *fsnd ;
            int         sound, point, x[5], y[5] ;
            int         max_pts ;
            double      max ;

            for (i = 0 ; i < number_inputs ; i++)
                        input[i] = 0.0 ;
            fsnd = fopen (word_number[pointer], "r") ;
            sound = 0 ;
            i = 0 ;
            flag = 0 ;
            while (flag != 1) {
                        fscanf (fsnd, "%f", &element) ;
                        if (feof(fsnd) !=0)
                                    flag = 1 ;
                        else if (i > 99)
                                    flag = 1 ;
                        else {
                                    in[i] = (double) element ;
                                    i++ ;
                                    }
                        if ((i == 15) && (flag == 0)) {
                                    f_mindist (f_map, in, &loc2[sound][0]) ;
                                    i = 0 ;
                                    sound++ ;
                                    }
                        }
            fclose (fsnd) ;
            for (i = 0 ; i < sound ; i++) {
                        loc3[i][0] = loc2[i][0] ;
                        loc3[i][1] = loc2[i][1] ;
                        }
            max_pts = sound ;

            ... Trajectory Reduction ...

            for (i = 0 ; i < j ; i++) {
                        point = loc2[i][0] + loc2[i][1] * f_xsize ;
```

```
                input[i] = point / 225.0 ;
                }
}


f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link outdat,ntraj,options_file/opt
/*
******************************** outdat.c ********************************

            Routines to create a path.dat file containing 75 point scalar
            trajectories from the first Kohonen net. Each scalar is from
            0 to 224. Short trajectories are filled with trailing 0's.

            The *.dat file is used to train a net quickly, eliminating file
            reading and running data through the first net repeatedly.
            *************************************************************

                                      Capt Gary Barmore, 3 Aug 88
*/

# include math
# include stdio
# include time

            float       map[20][20][225] ; /* output nodes */
            double      input[225] ; /* input nodes */
            double      gain, noise ;
            int         closest[2] ; /* closest node */
            int         xsize, ysize ; /* Size of array */
            int         number_inputs ;
            char        training_file[30], net_file[30], first_net_file[30] ;
            char        temp_file[15] ;
            char        net_name[15] ;

init (map)
            float       map[20][20][225] ;
{
            int         r, c, i ;
            FILE        *fnet ;

            fnet = fopen(net_file,"r") ;
            fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
            for (r = 0 ; r < ysize ; r++) {
                        for (c = 0 ; c < xsize ; c++) {
                                    for (i = 0 ; i < number_inputs ; i++) {
                                                fscanf (fnet,"%f", &map[r][c][i]) ;
                                                }
                                    }
                        }
            fclose (fnet) ;
}

mindist (map, inp, close)
            double                  inp[225] ;
            int                     close[2] ;
            float                   map[20][20][225] ;
{
            int                     r, c, i ;
            double                  dist ;
            double                  minimum = 999999.0 ;

            for (r = 0 ; r < ysize ; r++) {
                        for (c = 0 ; c < xsize ; c++) {
                                    dist = 0.0 ;
                                    for (i = 0 ; i < number_inputs ; i++)
```

```
                                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}

userinp ()
{
        int                     line ;
        int                     c ;
        struct tm               *localtime(), *time ;
        int                     *bintim ;

        do {
        printf ("OUTDAT: Create path.dat for net training ... \n\n") ;

                printf ("Enter name of header file containing words (less .hdr): ") ;
                scanf ("%s", temp_file) ;
                sprintf (training_file, "%s.hdr", temp_file) ;

                number_inputs = 75 ;

                printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
                scanf ("%s", net_name) ;
                sprintf (first_net_file, "%s.net", net_name) ;

                printf("Enter name of data file to create [less .dat]: ") ;
                scanf ("%s", net_name) ;
                sprintf (net_file, "%s.dat", net_name) ;

                printf("Ready to begin? (y/n) ") ;
                while ((c = getc (stdin)) == ' ' || c == '\n' || c == '\t')
                        ;
                } while (c != 'y') ;
}

save_net ()
{
        int                     r, c, i ;
        FILE                    *fnet ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long                    i ;
        char                s1[10] ;
        int                 ws_id = 1 ;
        int                 clear_flag = 1;
        FILE                *tf ;
        extern unsigned     _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        init (map) ; /* Initialize weights */
        read_trn_file () ;

        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
******************************** ntraj.c ********************************

        Routines to save 75 point scalar trajectoires from the first
        Kohonen net to save time in training the second Kohonen. Each
        scalar is from 0 to 224. Short trajectories are filled with
        trailing 0's.
        ***********************************************************
*/

# include math
# include stdio
# include stat

        double              innput[100][75] ;/* input vectors */

        extern double       input[225] ; /* input nodes */
        extern double       gain ;
        extern int          closest[2] ; /* closest node */
        extern int          neigh[2] ; /* neighbor */
        extern int          xsize, ysize ; /* Size of array */
        extern int          number_inputs ;
        extern char         training_file[30] ;
        extern char         first_net_file[30] ;
        extern char         net_file[30] ;

        int                 number_discretes ;
        int                 word_counter ;
        int                 num_words ;
        char                word_number[100][15] ;
        int                 f_ysize, f_xsize, f_number_inputs ;
        float               f_map[20][20][16] ;

read_trn_file ()
{
        FILE                *tf, *fnet ;
        int                 j, i, r, c ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        fnet = fopen (net_file, "w") ;
        fprintf (fnet, "%d\n", num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                read_word (i) ;
                for (j = 0 ; j < number_inputs ; j++) {
                        innput[i][j] = input[j] ;
                        fprintf (fnet, "%le\n", input[j]) ;
                        }
                }
        fclose (fnet) ;
}

getin ()
{
        int                     j ;

        if (word_counter == num_words)
                word_counter = 0 ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
        word_counter++ ;
}

get_rnd_in ()
{
        int                     i, j ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
}

weightem (map)
        float                   map[20][20][225] ;
{
        int                     nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
        if (nright >= xsize)
                nright = xsize - 1 ;
        nleft = closest[0] - neigh[0] + 1 ;
        if (nleft < 0)
                nleft = 0 ;
        nup = closest[1] - neigh[1] + 1 ;
        if (nup < 0)
                nup = 0 ;
        ndown = closest[1] + neigh[1] - 1 ;
        if (ndown >= ysize)
```

```
                ndown = ysize - 1 ;
        }
        else {
                nright = closest[0] ;
                nleft = closest[0] ;
                nup = closest[1] ;
                ndown = closest[1] ;
                }

        for (r = nup; r <= ndown ; r++) {
                for (c = nleft ; c <= nright ; c++) {
                        for (i = 0 ; i < number_inputs ; i++)
                                map[r][c][i] += gain * (input[i] - map[r][c][i]) ;
                        }
                }
}

read_word (pointer)
        int                     pointer ;
{
        int                     flag, r, c, i, j, k ;
        double                  in[16], d1, d2, d3, d4, d5 ;
        float                   element ;

        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int                     sound, point, x[5], y[5] ;
        int                     max_pts ;
        double                  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...
        for (i = 0 ; i < j ; i++) {
                point = loc2[i][0] + loc2[i][1] * f_xsize ;
                input[i] = point / 225.0 ;
                }
}


f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link twopic4,nplot,nprinter,mat3b,nweight8,options_file/opt
/*
******************************** twopic4.c ********************************

        Routine to show locations of words on second Kohonen net. The
        final version assumes a test set of 100 words. 10 each of zeros,
        ones, ... , nines (respectively). The graph identifies the last
        digit which lit it up.

        First Kohonen trajectories are compared by Euclidean distance
        algorithm. Trajectories are 75 scalar points long.

        The routine was written to test what nodes the training set will
        light up. Basically, this allows one to determine the way the
        training spread the inputs and the differentiability. It is not
        a test of the net.
        ***************************************************************
*/
# include math
# include stdio
# include time

        float   map[20][20][225] ; /* output nodes */
        double  input[225] ; /* input nodes */
        double  gain, noise ;
        double  mcount ;
        double  percent ;
        double  xoff = 0.0 ;
        double  yoff = 0.0 ;
        double  node_dist ;

        int     closest[2] ; /* closest node */
        int     neigh[2] ; /* neighbor */
        int     nrangex, nrangey ; /* neighbor range */
        int     nfactorx, nfactory ; /* neighbor factor */
        long    count ; /* # of iterations */
        int     graph ; /* # between plots */
        int     seed ;
        int     maxneighx, maxneighy ; /* Starting area */
        int     minneighx, minneighy ; /* Final area */
        int     xsize, ysize ; /* Size of array */
        int     number_inputs ;
        int     wrap_flag = 0 ;
        int     train_flag, train_discrete ;

        char    training_file[30], temp_file[30], first_net_file[30] ;
        char    net_file[30] ;

        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        extern int      xy[] ; /* array holding x,y */
        extern double   xdel, ydel ;
        extern double   xlow, xup, ylow, yup ;
        extern int      num_words ;
        extern char     word_number[100][15] ;

mindist (map, inp, close)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int     c ;

        printf ("\nTWOPIC4 (Plot Words for 0/1 Reduced Queued Traj)...\n") ;
                map_speech () ;
}

map_speech ()
{
        int     r, c, i, j, k ;
        char    sub_title[60], temp[30] ;
        char    name_trj[20] ;
        int     loc[125][2] ;
        FILE    *fnet, *flog ;
        int     sound ;
        short   length ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;

        read_trn_file () ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
```

```
          for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                              for (i = 0 ; i < number_inputs ; i++) {
                                        fscanf (fnet," %f", &map[r][c][i]) ;
                                        }
                              }
                    }
          fclose (fnet) ;

          sprintf (sub_title, "%s -> %s -> %s",
                    training_file, first_net_file, net_file) ;
          sprintf (name_trj, "%s.trj", temp) ;

          flog = fopen ("temp.log","w") ;
          fprintf (flog, "TWOPIC4: %s", name_trj) ;

          graph_test (training_file) ;
          length = (short) strlen (sub_title) ;
          draw_grid2 (ysize, xsize, sub_title, length) ;

          printf ("\nExpect %d calculations.\n", num_words) ;
          for (sound = 0 ; sound < num_words ; sound++) {
                    getin () ;
                    mindist (map, input, &loc[sound][0]) ;
                    printf ("%d : [%d,%d] dist = %le\n",
                              sound, loc[sound][0], loc[sound][1], node_dist) ;
                    fprintf (flog, "%d : [%d,%d] dist = %le\n",
                              sound, loc[sound][0], loc[sound][1], node_dist) ;
                    }
          printf ("\nCalculations finished.\n") ;
          fclose (flog) ;
          draw_speech_map (sound, loc) ;
          scanf ("%s",temp) ;
          clipoff () ;
          graphoff () ;
}


graph_test (name)
          char      name[30] ;
{
          char      title[79], labelx[79] ;
          float     xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
          float     yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
          int       points = 5 ;
          int       ws_id = 1 ;
          int       clear_flag = 1 ;
          short     length ;

          sprintf(title,"TWOPIC4: Kohonen TWO NETS -- %s", name) ;
          sprintf(labelx," ") ;
          graphon () ;
          gks$clear_ws (&ws_id, &clear_flag) ;
          gks$polyline (&points, xloc, yloc) ;
          prepcolmat (ysize, xsize) ;
          length = (short) strlen (title) ;
          outtitle (title, length) ;
          length = (short) strlen (labelx) ;
          outlabelx (labelx, length) ;
          clipon () ;
```

```
}

/*
********************************* mat3b.c *********************************

        Routines to perform graphics for net output. Note that the last
        version assumed second Kohonen net input is a set of 100 exemplars
        divided into 10 even classes (10 each). Thus the first class of
        ten is repreesented as 0, and the last as 9.
*/

# include stdio
# include math
# include <gksdefs.h>
# include <descrip.h>

# define BLACK0
# define WHITE1

        float   ptsx[20][20][5], ptsy[20][20][5] ;
        float   px[20][20], py[20][20] ;
        int     used[20][20] ;
        int     colmat[20][20] ;
        int     pattern[16]=
                {0, -15,-15, -12,-12, -11,-11, -1,-1, -2,-2, -4,-4, -5,-5, 1} ;

prepcolmat (ysize, xsize)
        int     xsize ;
        int     ysize ;
{
        int     r, c, xstart, ystart ;
        int     dx, dy ;

        dx = floor (550.0 / xsize) ;
        dy = floor (276.0 / ysize) ;

        xstart = 280 - dx * xsize / 2 ;
        ystart = 148 + dy * ysize / 2 ;

        for (c = 0 ; c < xsize ; c++) {
                for (r = 0 ; r < ysize ; r++) {
                        ptsx[r][c][4] = (ptsx[r][c][3] = (ptsx[r][c][0] =
                                xstart + c * dx)) ;
                        ptsx[r][c][2] = (ptsx[r][c][1] = ptsx[r][c][0] +dx- 1) ;

                        ptsy[r][c][4] = (ptsy[r][c][1] = (ptsy[r][c][0] =
                                ystart - r * dy)) ;
                        ptsy[r][c][3] = (ptsy[r][c][2] = ptsy[r][c][0] -dy+ 1) ;

                        px[r][c] = ptsx[r][c][0] + 8.0 ;
                        py[r][c] = ptsy[r][c][0] - 8.0 ;
                        }
                }
        for (r = 0 ; r < 20 ; r++) {
                for (c = 0 ; c < 20 ; c++) {
                        used[r][c] = 0 ;
                        }
                }
}
```

```
showem ()
{
        int     i ;
        float   rectx[16][6], recty[16][6] ;
        int     points = 5 ;

        for (i = 1 ; i < 16 ; i++) {
                rectx[i][4] = (rectx[i][3] = (rectx[i][0] = 173 + 16 * i)) ;
                rectx[i][2] = (rectx[i][1] = (rectx[i][0] + 15)) ;

                recty[i][4] = (recty[i][1] = (recty[i][0] = 30)) ;
                recty[i][3] = (recty[i][2] = 23) ;

                setfillstyle (pattern[i], i) ;
                gks$fill_area (&points, &rectx[i][0], &recty[i][0]) ;
                }
}

setfillstyle (pattern, pointer)
        int     pattern ;
        int     pointer ;
{
        int     style = 3 ;             /* hatch */
        int     color[16] = {0, 7,7, 3,3, 5,5, 4,4, 4, 6,6, 2,2, 1} ;

        if (pattern >= 0) {
                gks$set_fill_int_style (&pattern) ;
                }
        else {
                gks$set_fill_style_index (&pattern) ;
                gks$set_fill_int_style (&style) ;
                }
}

draw_net (number, loc)
        int     number, loc[64][2] ;
{
        int     j, i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] / 10.0 ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d",old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                j = i / 10.0 ;
                sprintf (s,"%3d",j) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i ;
                }
}

draw_neighbors (number, loc)
        int     number, loc[64][2] ;
{
        int     j, i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < 1 ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] / 10.0 ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                j = number / 10.0 ;
                sprintf (s,"%3d", j) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((number != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = number ;
                }
}

draw_speech_map (number, loc)
        int     number, loc[125][2] ;
{
        int     j, i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y, xx[2], yy[2] ;
        int     points = 2 ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] / 10.0 ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;
                j = i / 10.0 ;
                gks$set_text_color_index (&black) ;
                sprintf (s,"%3d", j) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i ;
                if (i != 0) {
                        xx[0] = px[loc[i-1][1]][loc[i-1][0]] ;
                        xx[1] = px[loc[i][1]][loc[i][0]] ;
                        yy[0] = py[loc[i-1][1]][loc[i-1][0]] ;
                        yy[1] = py[loc[i][1]][loc[i][0]] ;
/*
                        gks$polyline (&points, xx, yy) ;
*/
                        }
                }
}

draw_grid (ysize, xsize)
        int     xsize, ysize ;
{
        int     points = 5 ;
        int     r, c ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        gks$polyline (&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
}

draw_spectra (map, ysize, xsize)
        int     xsize, ysize ;
        float   map[20][20][16] ;
{
        int     points = 2 ;
        float   x[2], y[2] ;
        int     r, c, i ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        y[0] = ptsy[r][c][3] ;
                        for (i = 0 ; i < 15 ; i++) {
                                x[0] = ptsx[r][c][3] + 2.0 + (2.0 * i) ;
                                x[1] = x[0] ;
                                y[1] = y[0] + 16.0 * map[r][c][i] ;
                                gks$polyline (&points, x, y) ;
                                x[0] += 1.0 ;
                                x[1] += 1.0 ;
                                gks$polyline (&points, x, y) ;
                                }
                        }
                }
}
```

```
$ link twomask,options_file/opt
/*
********************************* twomask.c **********************************

          Routine to create *.msk file from *.net file. The *.msk file is an
          array of integers (mask[20][20]) corresponding to the nodes of a
          *.net file. Each integer is the number of weights which are != 0.

          Since weights represent scalar trajectories of 75 points, short
          trajectories are filled with trailing 0's. This obviously allows
          for some inaccuracy in determining the trajectory length since
          the first node is also represented as a 0. This will be corrected
          in later versions.
          ***************************; :*******************************
*/
# include math
# include stdio

          int       mask[20][20] ;
          float     map[20][20][225] ; /* output nodes */
          double    node_dist ;
          int       xsize, ysize ; /* Size of array */
          int       number_inputs ;
          char      training_file[30], temp_file[30], first_net_file[30] ;
          char      net_file[30] ;

non_zero (map)
          float     map[20][20][225] ;
{
          int       r, c, i, number ;

          for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                              number = 0 ;
                              for (i = 0 ; i < number_inputs ; i++) {
                                        printf ("%7.1e ", map[r][c][i]) ;
                                        if (map[r][c][i] > 5.0e-4)
                                                  number++ ;
                                        }
                              printf ("\n**** %d ****\n", number) ;
                              mask[r][c] = number ;
                              }
                    }
}

main ()
{
          printf ("\nTWOMASK (Creates net mask\n\n) ... ") ;
          find_mask () ;
}

find_mask ()
{
          int       r, c, i ;
          FILE      *fnet ;

          printf ("Enter name of output Koh net_file [less .net]: ") ;
          scanf ("%s", temp_file) ;
          sprintf (net_file, "%s.net", temp_file) ;
          sprintf (training_file, "%s.msk", temp_file) ;
```

```
        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %f", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        non_zero (map) ;
        save_mask () ;
}

save_mask ()
{
        FILE    *fmask ;
        int     r, c ;

        fmask = fopen (training_file, "w") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fprintf (fmask, "%d ", mask[r][c]) ;
                        }
                }
        fclose (fmask) ;
}
```

```
$ link twopic6,nwin5,lookup6,options_file/opt
/*
******************************** twopic6.c ********************************

        This routine identifies (w/o graphics) the node from a second Kohonen
        net which is closest in DTW distance to each of the digits in a
        specified set.

        In the original version, the trajectory inputs were 75 points long
        and scalar (1-225). In this version, the trajectories are 100
        points long.

        The DTW routine uses mask[] as the node length and length[] for the
        input length (i.e. number of trailing points not 0).
        ****************************************************************
*/
# include math
# include stdio
# include time

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        extern int      num_words ;
        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        extern int      f_xsize, f_ysize ;
        extern int      length[200] ;
        extern int      location[2000][2] ;

mindist (close)
        int             close[2] ;
{
        int             r, c, i ;
        double          dist ;
        double          distance ;
        double          minimum = 9.99e31 ;
        double          p1, p2 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dtw (&map2[r][c][0][0], location, mask[r][c],
                                length[0], &dist) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
        node_dist = minimum ;
```

```
}

main ()
{
          int          c ;

          printf ("\nTWOPIC6 (DTW Words for 0/1 Reduced Queued Traj)...\n") ;
                    map_speech () ;
}

map_speech ()
{
          int          r, c, i, j, k ;
          char         sub_title[60], temp[30] ;
          char         name_trj[20] ;
          int          loc[2] ;
          FILE         *fnet, *flog, *fmask ;
          int          sound ;
          short        length ;

          printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
          scanf ("%s", temp) ;
          sprintf (first_net_file, "%s.net", temp) ;

          printf ("Enter name of header file containing words (less .hdr): ") ;
          scanf ("%s", temp_file) ;
          sprintf (training_file, "%s.hdr", temp_file) ;

          printf ("Enter name of output Koh net_file [less .net]: ") ;
          scanf ("%s", temp_file) ;
          sprintf (net_file, "%s.net", temp_file) ;
          sprintf (temp, "%s.msk", temp_file) ;

          fnet = fopen (net_file, "r") ;
          fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
          for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                              for (i = 0 ; i < number_inputs ; i++) {
                                        fscanf (fnet," %f", &map[r][c][i]) ;
                                        }
                              }
                    }
          fclose (fnet) ;
          read_trn_file () ;
          fmask = fopen (temp, "r") ;
          for (r = 0 ; r < ysize ; r++) {
                    for (c = 0 ; c < xsize ; c++) {
                              fscanf (fmask,"%d", &mask[r][c]) ;
                              }
                    }
          fclose (fmask) ;

          flog = fopen ("temp.log","w") ;
          fprintf (flog, "TWOPIC6: %s\n", name_trj) ;
          fprintf (flog, "-> %s -> %s ->\n", first_net_file, net_file) ;
          fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

          printf ("\nExpect %d calculations.\n", num_words) ;
          fprintf (flog, "Expect %d calculations.\n", num_words) ;
          for (sound = 0 ; sound < num_words ; sound++) {
                getin () ;
                mindist (loc) ;
                k = loc[0] + loc[1] * xsize ;
                printf ("\n%d : [%d,%d] dist = %le ",
                        sound, loc[0], loc[1], node_dist) ;
                print_digit (k) ;
                fprintf (flog, "\n%d : [%d,%d] dist = %le ",
                        sound, loc[0], loc[1], node_dist) ;
                fprint_digit (k, flog) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int     template[200][2], utterance[200][2] ;
        int     t_length, u_length ;
        double  *ave_dist ;
{
        float   back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;

        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                        else if (d3 <= d2 && d3 < d1) {
                                                back_path[ptr][r] = d3 ;
                                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                                }
                                        else {
                                                back_path[ptr][r] = d1 ;
                                                b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                                }
                                        }
                                }
                        }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

/*
******************************** nwin5.c ********************************

        This routine supports TWOPIC6.C/EXE which finds the node closest
        in DTW(mask[],length[]) to a given digit.

        Trajectories are 100 (not 75 as in the first version) point scalars
        filled with trailing zeros.
        ****************************************************************
*/

# include math
# include stdio
# include stat

        extern double       input[225] ; /* input nodes */
        extern double       gain ;

        extern int          closest[2] ; /* closest node */
        extern int          neigh[2] ; /* neighbor */
        extern int          xsize, ysize ; /* Size of array */
        extern int          number_inputs ;
        extern int          train_discrete ;
        extern char         training_file[30] ;
        extern char         first_net_file[30] ;

        int                 number_discretes ;
        int                 word_counter ;
        int                 num_words ;
        char                word_number[200][15] ;
        int                 length[200] ;

        int                 f_ysize, f_xsize, f_number_inputs ;
        float               f_map[20][20][16] ;
        int                 location[2000][2] ;
        extern int          map2[20][20][100][2] ;
        extern float        map[20][20][225] ;

read_trn_file ()
{
        FILE               *tf, *fnet ;
        int                i, r, c, k, temp ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
```

```
                fscanf (tf, “%s”, word_number[i]) ;
        fclose (tf) ;
        word_counter = 0 ;

        fnet = fopen (first_net_file, “r”) ;
        fscanf (fnet,“%d %d %d”, &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet,“ %f”, &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                temp = (int) (map[r][c][i] * 226.0 - 1.0) ;
                                k = temp % f_xsize ;
                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                map2[r][c][i][0] = k ;
                                }
                        }
                }
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

get_rnd_in ()
{
        int             i ;
        double          max_rand = pow (2.0, 31.0) - 1.0 ;
        int             pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

read_word (pointer)
        int             pointer ;
{
        int             flag, r, c, i, j, k ;
        double          in[16], d1, d2, d3, d4, d5 ;
        float           element ;

        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        fsnd = fopen (word_number[pointer], “r”) ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, “%f”, &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 2000 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}


f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
/*
**************************** lookup6.c *************************************

          This is a look-up table that supports TWOPIC6*.C/EXE. Once
          nodes are identified (either with TWOPIC4* or TWOPIC6*), those
          assignments are stored in the table below.
*/
# include stdio

          int        look_up[100] = {9,9,9,3,7,7,7,6,6,6,
                                     9,9,3,3,3,3,6,6,6,6,
                                     9,9,3,2,3,0,6,6,6,6,
                                     1,1,1,3,3,0,3,6,6,6,
                                     1,1,1,3,0,0,0,4,4,4,
                                     3,1,9,3,0,0,0,0,4,4,
                                     5,3,3,3,3,3,7,0,4,0,
                                     5,5,2,3,2,7,7,7,8,2,
                                     5,5,2,2,3,3,7,4,8,8,
                                     5,5,9,3,0,3,4,3,8,8} ;

          char       digit[11][10] = {"zero",
                                     "one",
                                     "two",
                                     "three",
                                     "four",
                                     "five",
                                     "six",
                                     "seven",
                                     "eight",
                                     "nine",
                                     "noise"} ;

print_digit (node)
          int        node ;
{
          printf ("%s", digit[look_up[node]]) ;
}

fprint_digit (node, flog)
          int        node ;
          FILE       *flog ;
{
          fprintf (flog, "%s", digit[look_up[node]]) ;
}
```

```
$ link outdat3,ntraj3,options_file/opt
/*
******************************** outdat3.c ********************************

        This routine creates a set of stored trajectories in the file
        path.dat for use in training a second Kohonen net.

        Input trajectories are 100 point scalars (1-225) filled with
        trailing 0's.
        ************************************************************

                        Capt Gary Barmore, 25 Aug 88
*/

# include math
# include stdio
# include time

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */

        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], net_file[30], first_net_file[30] ;
        char            temp_file[15] ;
        char            net_name[15] ;

mindist (map, inp, close)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.9e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}

userinp ()
{
        int             line ;
        int             c ;
        struct tm       *localtime(), *time ;
        int             *bintim ;

        do {
                printf ("OUTDAT3: Prepare training data, second kohonen... \n\n") ;
                printf ("Enter name of header file containing words (less .hdr): ") ;
                scanf ("%s", temp_file) ;
                sprintf (training_file, "%s.hdr", temp_file) ;

                number_inputs = 100 ;

                printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
                scanf ("%s", net_name) ;
                sprintf (first_net_file, "%s.net", net_name) ;

                printf("Enter name of data file to create [less .dat]: ") ;
                scanf ("%s", net_name) ;
                sprintf (net_file, "%s.dat", net_name) ;

                printf("Ready to begin? (y/n) ") ;
                while ((c = getc (stdin)) = = ' ' || c = = '\n' || c = = '\t')
                        ;
                } while (c != 'y') ;
}

main()
{
        extern unsigned    _stklen ;
        _stklen = 8192 ;

        userinp () ; /* Get input values */
        read_trn_file () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
********************************* ntraj3.c ********************************

        This routine supports OUTDAT3.C/EXE in creating a set of stored
        trajectories (path.dat) for training a second Kohonen net.

        Trajectories are 100 point scalars (0-225) filled with trailing 0's.
        ************************************************************
                        G. Barmore25 Aug 88
*/

# include math
# include stdio
# include stat

        double          innput[100][100] ;/* input vectors */

        extern double   input[225] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;
        extern char     net_file[30] ;
        int                     number_discretes ;
        int                     word_counter ;
        int                     num_words ;
        char                    word_number[100][15] ;

        int                     f_ysize, f_xsize, f_number_inputs ;
        float                   f_map[20][20][16] ;

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     j, i, r, c ;

        tf = fopen (training_file, “r”) ;
        fscanf (tf, “%d”, &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, “%s”, word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, “r”) ;
        fscanf (fnet,”%d %d %d”, &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet,” %f”, &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        fnet = fopen (net_file, “w”) ;
        fprintf (fnet, “%d\n”, num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                read_word (i) ;
                for (j = 0 ; j < number_inputs ; j++) {
                        innput[i][j] = input[j] ;
                        fprintf (fnet, “%le\n”, input[j]) ;
                        }
                }
        fclose (fnet) ;
}

read_word (pointer)
        int                     pointer ;
{
        int                     flag, r, c, i, j, k ;
        double                  in[16], d1, d2, d3, d4, d5 ;
        float                   element ;
        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int                     sound, point, x[5], y[5] ;
        int                     max_pts ;
        double                  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], “r”) ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                point = 1 + loc2[i][0] + loc2[i][1] * f_xsize ;
                input[i] = point / 226.0 ;
                }
}


f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link twobas3,nweight11,options_file/opt
/*
******************************** twobas3.c ********************************

        These routines train a second Kohonen net to process 100 point
        scalar trajectories filled with trailing 0's.

        Initial weights are random. Stored training trajectories are
        found in a path.dat file. Conscience is an user supplied variable
        in this version.
        ************************************************************

                                Capt Gary Barmore, 1 Sep 88
*/

# include math
# include stdio
# include time

        int     conscience[20][20] ;/* records # times closest */
        int     nodes ;                          /* number of nodes */
        double  consc = 1.1 ;             /* conscience factor */
        float   map[20][20][225] ; /* output nodes */
        double  input[225] ; /* input nodes */
        double  gain, noise ;
        double  mcount ;
        double  percent ;
        double  xoff = 0.0 ;
        double  yoff = 0.0 ;

        int     closest[2] ; /* closest node */
        int     neigh[2] ; /* neighbor */
        int     nrangex, nrangey ; /* neighbor range */
        int     nfactorx, nfactory ; /* neighbor factor */
        long    count ; /* # of iterations */
        int     graph ; /* # between plots */
        int     seed ;
        int     maxneighx, maxneighy ; /* Starting area */
        int     minneighx, minneighy ; /* Final area */
        int     xsize, ysize ; /* Size of array */
        int     number_inputs ;
        int     wrap_flag = 0 ;
        int     train_flag, train_discrete ;

        char    training_file[30], net_file[30], first_net_file[30] ;
        char    temp_file[15] ;
        char    net_name[15] ;

        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        struct flg {
                int             rnd_in ;
                } flag ;
        extern int              xy[] ; /* array holding x,y */
        extern double           xdel, ydel ;
        extern double           xlow, xup, ylow  yup ;
        extern int              tr_length ;

init (map)
        float                   map[20][20][225] ;
{
        int                     r, c, i ;
        float                   max_rand = pow(2.0, 31.0) - 1.0 ;

        nodes = ysize * xsize ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        conscience[r][c] = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                map[r][c][i] = rand () / max_rand ;
                                }
                        }
                }
}

mindist (map, inp, close, its)
        double                  inp[225] ;
        int                     close[2] ;
        float                   map[20][20][225] ;
        long                    its ;
{
        int                     r, c, i ;
        double                  dist ;
        double                  minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        if (conscience[r][c] < consc * its / nodes ) {
                                for (i = 0 ; i < number_inputs ; i++)
                                        dist += pow(inp[i] - map[r][c][i], 2.0);
                                if (dist < minimum) {
                                        minimum = dist ;
                                        close[0] = c ;
                                        close[1] = r ;
                                        }
                                }
                        }
                }
        conscience[close[1]][close[0]] += 1 ;
}

userinp ()
{
        int                     line ;
        int                     c ;

        do {
                printf ("TWOBAS2: TWO KOHonen net training (output only)... \n\n") ;

                printf ("Enter conscience factor (> 1.0): [float] ") ;
                scanf ("%f", &consc) ;
                if (consc < 1.0)
        consc = 1.5 ;

printf("Enter size 'm n' (for an m x n) of array = ? [int int] ") ;
scanf("%d %d", &ysize, &xsize) ;
if (ysize < 2)
        ysize = 2 ;
else if (ysize > 20)
        ysize = 20 ;
if (xsize < 2)
        xsize = 2 ;
else if (xsize > 20)
        xsize = 20 ;

printf ("Do you want 0) sequential training,\n") ;
printf (" 1) randomized training? ") ;
scanf ("%d", &flag.rnd_in) ;

printf ("Enter name of header file containing words (less .hdr): ") ;
scanf ("%s", temp_file) ;
sprintf (training_file, "%s.hdr", temp_file) ;

train_discrete = 1 ;
number_inputs = 100 ;

printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
scanf ("%s", net_name) ;
sprintf (first_net_file, "%s.net", net_name) ;

printf("Enter name of net file to create [less .net]: ") ;
scanf ("%s", net_name) ;
sprintf (net_file, "%s.net", net_name) ;

printf ("Number of iterations = ? [int] ") ;
scanf ("%ld", &count) ;
if (count < = 10 || count > 200000)
        count = 100 ;
mcount = (double) count ;

printf ("Number of iterations between status messages = ? [int] ") ;
scanf ("%d", &graph) ;
if (graph < 1 || graph > count)
        graph = 10 ;

ingain () ;

printf ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &maxneighy, &maxneighx) ;
if (maxneighx < 2 || maxneighx > xsize - 1)
        maxneighx = 2 ;
if (maxneighy < 2 || maxneighy > ysize - 1)
        maxneighy = 2 ;

printf ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &minneighy, &minneighx) ;
if (minneighx < 1 || minneighx > maxneighx)
        minneighx = 1 ;
if (minneighy < 1 || minneighy > maxneighy)
        minneighy = 1 ,

printf
```

```
                        ("Initial seed for random # generator = ? [int] ");
                scanf ("%d", &seed) ;
                if (seed == 0) {
                        seed = 138 ;
                        }
                srand (seed) ;

                wrap_flag = 0 ;

                printf("Ready to begin? (y/n) ") ;
                while ((c = getc (stdin)) == ' ' || c == '\n' || c == '\t')
                        ;
                } while (c != 'y') ;
}

ingain ()
{
        int             line ;

        printf("For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR : ");
        scanf ("%d", &gcurve.type) ;

        if (gcurve.type == 0 || gcurve.type == 1) {
                printf ("Maximum gain = ? [float] ") ;
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ;

                printf ("Minimum gain = ? [float] ") ;
                scanf ("%E", &gcurve.mingain) ;
                if (gcurve.mingain <= 0.0 || gcurve.mingain >= 1.0)
                        gcurve.mingain = 0.0 ;
                }
        else {
                printf ("First segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.maxgain) ;
                if (gcurve.maxgain >= 1.0 || gcurve.maxgain <= 0.0)
                        gcurve.maxgain = .99 ;

                printf ("Second segment starting gain = ? [float] ") ;
                scanf ("%E", &gcurve.midgain) ;
                if (gcurve.midgain <= 0.0 || gcurve.midgain >= 1.0)
                        gcurve.midgain = 0.0 ;

                printf ("Second segment starting iteration = ? [float] ") ;
                scanf ("%d", &gcurve.midtime) ;
                if (gcurve.midtime <= 0 || gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long            i ;
{
        if (gcurve.type == 0)
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) +
                        gcurve.mingain ;
        else if (gcurve.type == 1)
```

```
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i -
                        count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
                }
}

save_net ()
{
        int                     r, c, i ;
        FILE                    *fnet ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fprintf (fnet," %f", map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long                    i ;
        char                    s1[10] ;
        int                     ws_id = 1 ;
        int                     clear_flag = 1;
        FILE                    *tf ;
        extern unsigned         _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */
        read_trn_file () ;

        for (i = 1 ; i <= count ; i++) {
                if (i % graph == 0) {
                        printf ("TWOBAS3: gain = %f, yrange = %d, ",
                                gain, nrangey) ;
                        printf ("xrange = %d, iteration # %d", nrangex,i) ;
                        printf (" (of %ld)\n", count) ;
                        if (access (net_file,0) == 0)
                                delete (net_file) ;
                        save_net () ;
                        }
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in == 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (map, input, closest, i) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                                ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
******************************** nweight11.c ********************************

        These routines support TWOBAS3.C/EXE in training a second Kohonen
        net with 100 point scalar inputs filled with trailing 0's. It
        uses stored training inputs in a path.dat file.
        ****************************************************************
                G. BARMORE25 AUG 88
*/

# include math
# include stdio
# include stat

        double          innput[100][100] ;/* input vectors */

        extern double   input[225] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

read_trn_file ()
{
        FILE            *tf, *fnet ;
        int                     j, i, r, c ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        fnet = fopen ("path.dat", "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                for (j = 0 ; j < number_inputs ; j++)
                        fscanf (fnet, "%le", &innput[i][j]) ;
                }
        fclose (fnet) ;
        word_counter = 0 ;
}

getin ()
{
        int                     j ;

        if (word_counter == num_words)
                word_counter = 0 ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
        word_counter++ ;
}

get_rnd_in ()
{
        int                     i, j ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        for (j = 0 ; j < number_inputs ; j++)
                input[j] = innput[word_counter][j] ;
}

weightem (map)
        float                   map[20][20][225] ;
{
        int                     nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
                if (nright >= xsize)
                        nright = xsize - 1 ;
                nleft = closest[0] - neigh[0] + 1 ;
                if (nleft < 0)
                        nleft = 0 ;
                nup = closest[1] - neigh[1] + 1 ;
                if (nup < 0)
                        nup = 0 ;
                ndown = closest[1] + neigh[1] - 1 ;
                if (ndown >= ysize)
                        ndown = ysize - 1 ;
                }
        else {
                nright = closest[0] ;
                nleft = closest[0] ;
                nup = closest[1] ;
                ndown = closest[1] ;
                }

        for (r = nup; r <= ndown ; r++) {
                for (c = nleft ; c <= nright ; c++) {
                        for (i = 0 ; i < number_inputs ; i++)
                                map[r][c][i] += gain * (input[i] - map[r][c][i]) ;
                        }
                }
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;

        int     loc2[2000][2], loc3[2000][2] ;
        FILE    *fsnd ;
        int     sound, point, x[5], y[5] ;
        int     max_pts ;
        double  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                point = 1 + loc2[i][0] + loc2[i][1] * f_xsize ;
                input[i] = point / 226.0 ;
                }
}


f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link twopic4b,nplot,nprinter,mat3,nweight12,options_file/opt
/*
******************************** twopic4b.c ********************************

        This routine is used to show graphically those nodes which 'light
        up' using Euclidean distance when the training set of inputs is
        applied to a second Kohonen net.

        Inputs are 100 point scalar (1-225) trajectories filled with
        trailing 0's.
        ************************************************************

*/
# include math
# include stdio
# include time

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          gain, noise ;
        double          mcount ;
        double          percent ;
        double          xoff = 0.0 ;
        double          yoff = 0.0 ;
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             neigh[2] ; /* neighbor */
        int             nrangex, nrangey ; /* neighbor range */
        int             nfactorx, nfactory ; /* neighbor factor */
        long            count ; /* # of iterations */
        int             graph ; /* # between plots */
        int             seed ;
        int             maxneighx, maxneighy ; /* Starting area */
        int             minneighx, minneighy ; /* Final area */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;
        int             wrap_flag = 0 ;
        int             train_flag, train_discrete ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        extern int      xy[] ; /* array holding x,y */
        extern double   xdel, ydel ;
        extern double   xlow, xup, ylow, yup ;
        extern int      num_words ;
        extern char     word_number[100][15] ;

 mindist (map, inp, close)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
```

```
{
        int         r, c, i ;
        double      dist ;
        double      minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int         c ;

        printf ("\nTWOPIC4b (Plot Words for 0/1 Reduced Queued Traj)...\n") ;
                map_speech () ;
}

map_speech ()
{
        int         r, c, i, j, k ;
        char        sub_title[60], temp[30] ;
        char        name_trj[20] ;
        int         loc[125][2] ;
        FILE        *fnet, *flog ;
        int         sound ;
        short       length ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;

        read_trn_file () ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %f", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        sprintf (sub_title, "%s -> %s ->", first_net_file, net_file) ;
        sprintf (name_trj, "%s.trj", temp) ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "TWOPIC4b: %s", name_trj) ;

        graph_test (training_file) ;
        length = (short) strlen (sub_title) ;
        draw_grid2 (ysize, xsize, sub_title, length) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        for (sound = 0 ; sound < num_words ; sound++) {
                getin () ;
                mindist (map, input, &loc[sound][0]) ;
                printf ("%d : [%d,%d] dist = %le\n",
                        sound, loc[sound][0], loc[sound][1], node_dist) ;
                fprintf (flog, "%d : [%d,%d] dist = %le\n",
                        sound, loc[sound][0], loc[sound][1], node_dist) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
        draw_speech_map (sound, loc) ;
        scanf ("%s",temp) ;
        clipoff () ;
        graphoff () ;
}

graph_test (name)
        char    name[30] ;
{
char    title[79], labelx[79] ;
        float   xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
        float   yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
        int     points = 5 ;
        int     ws_id = 1 ;
        int     clear_flag = 1 ;
        short   length ;

        sprintf(title,"TWOPIC4b: Kohonen TWO NETS -- %s", name) ;
        sprintf(labelx," ") ;
        graphon () ;
        gks$clear_ws (&ws_id, &clear_flag) ;
        gks$polyline (&points, xloc, yloc) ;
        prepcolmat (ysize, xsize) ;
        length = (short) strlen (title) ;
        outtitle (title, length) ;
        length = (short) strlen (labelx) ;
        outlabelx (labelx, length) ;
        clipon () ;
}

/*
********************************* mat3.c *************************************

        These routines support graphics operations in showing which node in
        a second Kohonen net lights up when the training set of trajectories
        is applied.
        Trajectories are 100 point scalars with trailing 0's.
*/

# include stdio
# include math
# include <gksdefs.h>
# include <descrip.h>

# define BLACK0
# define WHITE1

        float   ptsx[20][20][5], ptsy[20][20][5] ;
        float   px[20][20], py[20][20] ;
        int     used[20][20] ;
        int     colmat[20][20] ;
        int     pattern[16] =
                {0, -15,-15, -12,-12, -11,-11, -1,-1, -2,-2, -4,-4, -5,-5, 1} ;

prepcolmat (ysize, xsize)
        int     xsize ;
        int     ysize ;
{
        int     r, c, xstart, ystart ;
        int     dx, dy ;

        dx = floor (550.0 / xsize) ;
        dy = floor (276.0 / ysize) ;

        xstart = 280 - dx * xsize / 2 ;
        ystart = 148 + dy * ysize / 2 ;

        for (c = 0 ; c < xsize ; c++) {
                for (r = 0 ; r < ysize ; r++) {
                        ptsx[r][c][4] = (ptsx[r][c][3] = (ptsx[r][c][0] =
                                xstart + c * dx)) ;
                        ptsx[r][c][2] = (ptsx[r][c][1] = ptsx[r][c][0] +dx- 1) ;

                        ptsy[r][c][4] = (ptsy[r][c][1] = (ptsy[r][c][0] =
                                ystart - r * dy)) ;
                        ptsy[r][c][3] = (ptsy[r][c][2] = ptsy[r][c][0] -dy+ 1) ;

                        px[r][c] = ptsx[r][c][0] + 8.0 ;
                        py[r][c] = ptsy[r][c][0] - 8.0 ;
                        }
                }
        for (r = 0 ; r < 20 ; r++) {
                for (c = 0 ; c < 20 ; c++) {
                        used[r][c] = 0 ;
                        }
                }
}

showem ()
{
        int     i ;
        float   rectx[16][6], recty[16][6] ;
        int     points = 5 ;

        for (i = 1 ; i < 16 ; i++) {
                rectx[i][4] = (rectx[i][3] = (rectx[i][0] = 173 + 16 * i)) ;
                rectx[i][2] = (rectx[i][1] = (rectx[i][0] + 15)) ;

                recty[i][4] = (recty[i][1] = (recty[i][0] = 30)) ;
                recty[i][3] = (recty[i][2] = 23) ;

                setfillstyle (pattern[i], i) ;
                gks$fill_area (&points, &rectx[i][0], &recty[i][0]) ;
                }
}

setfillstyle (pattern, pointer)
        int     pattern ;
        int     pointer ;
{
        int     style = 3 ;             /* hatch */
        int     color[16] = {0, 7,7, 3,3, 5,5, 4,4, 4, 6,6, 2,2, 1} ;

        if (pattern >= 0) {
                gks$set_fill_int_style (&pattern) ;
                }
        else {
                gks$set_fill_style_index (&pattern) ;
                gks$set_fill_int_style (&style) ;
                }
}

draw_net (number, loc)
        int     number, loc[64][2] ;
{
        int     i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d",-old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                sprintf (s,"%3d",i+1) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i+1 ;
                }
}

draw_neighbors (number, loc)
```

```
        int     number, loc[64][2] ;
{
        int     i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < 1 ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                sprintf (s,"%3d", (number+1)) ;
                gks$set_text_color_index (&black) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((number+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = number+1 ;
                }
}

draw_speech_map (number, loc)
        int     number, loc[125][2] ;
{
        int     i, old_value ;
        int     white = 0 ;
        int     black = 1 ;
        float   x, y, xx[2], yy[2] ;
        int     points = 2 ;
        char    s[4] ;
        $DESCRIPTOR(s_dsc,s) ;

        for (i = 0 ; i < number ; i++) {
                x = ptsx[loc[i][1]][loc[i][0]][0] + 4.0 ;
                y = ptsy[loc[i][1]][loc[i][0]][0] - 4.0 ;
                old_value = used[loc[i][1]][loc[i][0]] ;

                gks$set_text_color_index (&white) ;
                sprintf (s,"%3d", old_value) ;
                gks$text (&x, &y, &s_dsc) ;

                gks$set_text_color_index (&black) ;
                sprintf (s,"%3d", i+1) ;
                gks$text (&x, &y, &s_dsc) ;

                if ((i+1 != old_value) && (old_value != 0)) {
                        y -= 6.0 ;
                        sprintf (s,"***") ;
                        gks$text (&x, &y, &s_dsc) ;
                        }
                used[loc[i][1]][loc[i][0]] = i+1 ;

                if (i != 0) {
                        xx[0] = px[loc[i-1][1]][loc[i-1][0]] ;
                        xx[1] = px[loc[i][1]][loc[i][0]] ;
                        yy[0] = py[loc[i-1][1]][loc[i-1][0]] ;
                        yy[1] = py[loc[i][1]][loc[i][0]] ;
/*
                        gks$polyline (&points, xx, yy) ;
*/
                        }
                }
}

draw_grid (ysize, xsize)
        int     xsize, ysize ;
{
        int     points = 5 ;
        int     r, c ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        gks$polyline (&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
}
draw_spectra (map, ysize, xsize)
        int     xsize, ysize ;
        float   map[20][20][16] ;
{
        int     points = 2 ;
        float   x[2], y[2] ;
        int     r, c, i ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        y[0] = ptsy[r][c][3] ;
                        for (i = 0 ; i < 15 ; i++) {
                                x[0] = ptsx[r][c][3] + 2.0 + (2.0 * i) ;
                                x[1] = x[0] ;
                                y[1] = y[0] + 16.0 * map[r][c][i] ;
                                gks$polyline (&points, x, y) ;
                                x[0] += 1.0 ;
                                x[1] += 1.0 ;
                                gks$polyline (&points, x, y) ;
                                }
                        }
                }
}

draw_grid2 (ysize, xsize, sub_title, length)
        int     xsize, ysize ;
        char    sub_title[30] ;
        short   length ;
{
        int     points = 5 ;
        int     r, c ;
        float   xloc, yloc ;
        struct dsc$descriptor title_dsc = { length,
                                                DSC$K_DTYPE_T,
                                        DSC$K_CLASS_S,
                                        sub_title } ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        gks$polyline (&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
        xloc = 277.0 - 3.0 * length ;
        yloc = 2.0 ;
        gks$text (&xloc, &yloc, &title_dsc) ;
}

statusem (gain, nrangey, nrangex, its)
        double  gain ;
        int     nrangey, nrangex ;
        long    its ;
{
        float   xloc, yloc ;
        char    s[60] ;
        $DESCRIPTOR(s_dsc,s) ;

        sprintf (s,
                "Gain = %4.2f Neighbors = %2d,%2d Iteration # %5ld",
        gain, nrangey, nrangex, its) ;
        xloc = 76.0 ;
        yloc = 2.0 ;
        gks$text (&xloc, &yloc, &s_dsc) ;
}

colorem (ysize, xsize)
        int     xsize, ysize ;
{
        int     r, c, color ;
        int     points = 5 ;

        for (r = 0 ; r < ysize ; r   +) {
                for (c = 0 ; c < xsize ; c++) {
                        color = colmat[r][c] ;
                        setfillstyle (pattern[color], color) ;
                        gks$fill_area(&points, &ptsx[r][c][0], &ptsy[r][c][0]);
                        }
                }
}

pickcolors ()
{
        return ;
}

/*
********************************** nweight12.c **********************************

        These routines support TWOPIC4B.C/EXE in showing which nodes of
        a Kohonen net light up when the training set of trajectories is
        applied.

        Trajectories are 100 point scalar (1-225) inputs filled with
        trailing 0's.
        ***************************************************************
```

```
*/

# include math
# include stdio
# include stat

          extern double      input[225] ; /* input nodes */
          extern double      gain ;

          extern int         closest[2] ; /* closest node */
          extern int         neigh[2] ; /* neighbor */
          extern int         xsize, ysize ; /* Size of array */
          extern int         number_inputs ;
          extern int         train_discrete ;
          extern char        training_file[30] ;
          extern char        first_net_file[30] ;

          int                number_discretes ;
          int                word_counter ;
          int                num_words ;
          char               word_number[100][15] ;

          int                f_ysize, f_xsize, f_number_inputs ;
          float              f_map[20][20][16] ;

read_trn_file ()
{
          FILE               *tf, *fnet ;
          int                i, r, c ;

          tf = fopen (training_file, "r") ;
          fscanf (tf, "%d", &num_words) ;
          for (i = 0 ; i < num_words ; i++)
                    fscanf (tf, "%s", word_number[i]) ;
          fclose (tf) ;
          word_counter = 0 ;

          fnet = fopen (first_net_file, "r") ;
          fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
          for (r = 0 ; r < f_ysize ; r++) {
                    for (c = 0 ; c < f_xsize ; c++) {
                              for (i = 0 ; i < f_number_inputs ; i++) {
                                        fscanf (fnet," %f", &f_map[r][c][i]) ;
                                        }
                              }
                    }
          fclose (fnet) ;
}

getin ()
{
          if (word_counter == num_words)
                    word_counter = 0 ;
          read_word (word_counter) ;
          word_counter++ ;
}

get_rnd_in ()
{
          int                i ;
```

```
        double          max_rand = pow (2.0, 31.0) - 1.0 ;
        int             pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

weightem (map)
        float           map[20][20][225] ;
{
        int             nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
                if (nright >= xsize)
                        nright = xsize - 1 ;
                nleft = closest[0] - neigh[0] + 1 ;
                if (nleft < 0)
                        nleft = 0 ;
                nup = closest[1] - neigh[1] + 1 ;
                if (nup < 0)
                        nup = 0 ;
                ndown = closest[1] + neigh[1] - 1 ;
                if (ndown >= ysize)
                        ndown = ysize - 1 ;
                }
        else {
                nright = closest[0] ;
                nleft = closest[0] ;
                nup = closest[1] ;
                ndown = closest[1] ;
                }

        for (r = nup; r <= ndown ; r++) {
                for (c = nleft ; c <= nright ; c++) {
                        for (i = 0 ; i < number_inputs ; i++)
                                map[r][c][i] += gain * (input[i] - map[r][c][i]) ;
                        }
                }
}

read_word (pointer)
        int             pointer ;
{
        int             flag, r, c, i, j, k ;
        double          in[16], d1, d2, d3, d4, d5 ;
        float           element ;

        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fend, "%f", &element) ;
                if (feof(fend) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fend) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                point = 1 + loc2[i][0] + loc2[i][1] * f_xsize ;
                input[i] = point / 226.0 ;
                }
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link twopic8,nwin6,options_file/opt
/*
******************************** twopic8.c ********************************

        These routines find the closest digit, from a specified (usually non-
        training) set of digits, to each node in a second Kohonen net.
        This is used to identify nodes for later use of the net in recognizing
        unknown digits.

        Inputs to the net are 100 point scalar (1-225) trajectories filled
        with trailing 0's.

        The 'closest' process uses DTW(mask[],length[]) distance.
        ************************************************************
*/
# include math
# include stdio
# include time

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        extern int      num_words ;
        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        extern int      f_xsize, f_ysize ;

        extern int      location[2000][2] ;
        int             innput[200][100][2] ;
        extern int      length[200] ;
        extern char     word_number[200][15] ;

mindist (r, c, close)
        int             r, c ;
        int             *close ;
{
        int             sound ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (sound = 0 ; sound < num_words ; sound++) {
                dtw (&map2[r][c][0][0], &innput[sound][0][0], mask[r][c],
                        length[sound], &dist) ;
                if (dist < minimum) {
                        minimum = dist ;
                        *close = sound ;
                        }
                }
        node_dist = minimum ;
}
```

```
main ()
{
        int         c ;

        printf ("\nTWOPIC8 (Gives closest word for each node: 100 wts)...\n") ;
        map_speech () ;
}

map_speech ()
{
        int         r, c, i, j, k ;
        char        name_trj[30], temp[30] ;
        int         loc ;
        FILE        *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_fiie) ;
        sprintf (temp, "%s.msk", temp_file) ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %f", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        read_trn_file () ;

        fmask = fopen (temp, "r") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fscanf (fmask,"%d", &mask[r][c]) ;
                        }
                }
        fclose (fmask) ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "TWOPIC8: %s\n", name_trj) ;
        fprintf (flog, "-> %s -> %s ->\n", first_net_file, net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        printf ("\nReading word: ") ;
        for (r = 0 ; r < num_words ; r++) {
                printf ("%d ", r) ;
                getin () ;
                for (c = 0 ; c < number_inputs ; c++) {
                        innput[r][c][0] = location[c][0] ;
                        innput[r][c][1] = location[c][1] ;
                        }
                }
        printf ("\n") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        mindist (r, c, &loc) ;
                        k = c + r * xsize ;
                        printf ("\nNode %3d : word # %3d, dist = %le ",
                                k, loc, node_dist) ;
                        printf ("(%s)", word_number[loc]) ;
                        fprintf (flog, "\nNode %3d : word # %3d, dist = %le ",
                                k, loc, node_dist) ;
                        fprintf (flog, "(%s)", word_number[loc]) ;
                        }
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int     template[200][2], utterance[200][2] ;
        int     t_length, u_length ;
        double  *ave_dist ;
{
        float   back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;

        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

/*
******************************** nwin6.c ********************************

        These routines support TWOPIC8.C/EXE in finding the closest digit
        to each node in a second Kohonen net.

        Inputs are 100 point scalar (1-225) trajectories filled with
        trailing 0's.

        The distance routine in DTW(mask[],length[]).
        ************************************************************
                                        G. Barmore25 Aug 88
*/

# include math
# include stdio
# include stat

        extern double   input[225] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[200][15] ;
        int             length[200] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;
        int             location[2000][2] ;
        extern int      map2[20][20][100][2] ;
        extern float    map[20][20][225] ;

read_trn_file ()
{
        FILE            *tf, *fnet ;
        int             i, r, c, k, temp ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;
        word_counter = 0 ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                temp = (int) (map[r][c][i] * 226.0 - 1.0) ;
                                k = temp % f_xsize ;
                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                map2[r][c][i][0] = k ;
                                }
                        }
                }
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

get_rnd_in ()
{
        int             i ;
        double          max_rand = pow (2.0, 31.0) - 1.0 ;
        int             pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

read_word (pointer)
        int             pointer ;
{
        int             flag, r, c, i, j, k ;
        double          in[16], d1, d2, d3, d4, d5 ;
        float           element ;

        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[pointer] = j ;
        for (i = j ; i < 2000 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link outdat4,options_file/opt
/*
******************************** outdat4.c ********************************

        This routine creates a *.dat file containing stored trajectories
        to train second Kohonen nets.

        Trajectories are 100 x-y pairs filled with trailing -1's.
        **************************************************************

                                Capt Gary ʟ armore, 2 Sep 88
*/

# include math
# include stdio
# include time

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */

        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], net_file[30], first_net_file[30] ;
        char            temp_file[15] ;
        char            net_name[15] ;
        double          innput[100][100] ;/* input vectors */
        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;
        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;
        int             location[2000][2] ;

mindist (map, inp, close)
        double          inp[225] ;
        int             close[2] ;
        float           map[20][20][225] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.9e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dist += pow (inp[i] - map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}

userinp ()
{
        int                     line ;
        int                     c ;
        struct tm               *localtime(), *time ;
        int                     *bintim ;

        do {
                printf ("OUTDAT4: Prepare training data [x,y], second kohonen... \n\n") ;

                printf ("Enter name of header file containing words (less .hdr): ") ;
                scanf ("%s", temp_file) ;
                sprintf (training_file, "%s.hdr", temp_file) ;

                number_inputs = 100 ;

                printf ("Enter name of pre-processor Kohonen net file [less .net]: ") ;
                scanf ("%s", net_name) ;
                sprintf (first_net_file, "%s.net", net_name) ;

                printf("Enter name of data file to create [less .dat]: ") ;
                scanf ("%s", net_name) ;
                sprintf (net_file, "%s.dat", net_name) ;

                printf("Ready to begin? (y/n) ") ;
                while ((c = getc (stdin)) == ' ' || c == '\n' || c == '\t')
                        ;
                } while (c != 'y') ;
}

main()
{
        extern unsigned    _stklen ;
        _stklen = 8192 ;

        userinp () ; /* Get input values */
        printf ("\n") ;
        read_trn_file () ;
        printf ("\n.DAT file: %s saved!\n", net_file) ;
}

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     j, i, r, c ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
        fnet = fopen (net_file, "w") ;
        fprintf (fnet, "%d\n", num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                printf (" %d", i) ;
                read_word (i) ;
                for (j = 0 ; j < number_inputs ; j++) {
                        fprintf (fnet, "%d %d\n",
                                location[j][0], location[j][1]) ;
                        }
                }
        fclose (fnet) ;
}

read_word (pointer)
        int                     pointer ;
{
        int                     flag, r, c, i, j, k ;
        double                  in[16], d1, d2, d3, d4, d5 ;
        float                   element ;

        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int                     sound, point, x[5], y[5] ;
        int                     max_pts ;
        double                  max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        for (i = j ; i < number_inputs ; i++)
                location[i][0] = (location[i][1] = -1) ;
}


f_mindist (f_map, inp, close)
        double                  inp[16] ;
        int                     close[2] ;
        float                   f_map[20][20][16] ;
{
        int                     r, c, i ;
        double                  dist ;
        double                  minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
******************************** twobas4.c ********************************
        Routine to train a second Kohonen net using stored trajectories
        in a *.dat file (created with OUTDAT4.EXE).

        Trajectories are 100 x-y pairs filled with trailing -1. This
        version includes conscience.
        ************************************************************

Capt Gary Barmore, 7 Sep 88
*/

# include math
# include stdio
# include time

        int             conscience[20][20] ;/* records # times closest */
        int             nodes ;                         /* number of nodes */
        double          consc = 1.1 ;           /* conscience factor */
        float           map[20][20][100][2] ; /* output nodes */
        double          input[100][2] ; /* input nodes */
        double          gain, noise ;
        double          mcount ;
        double          percent ;
        double          xoff = 0.0 ;
        double          yoff = 0.0 ;

        int             closest[2] ; /* closest node */
        int             neigh[2] ; /* neighbor */
        int             nrangex, nrangey ; /* neighbor range */
        int             nfactorx, nfactory ; /* neighbor factor */
        long            count ; /* # of iterations */
        int             graph ; /* # between plots */
        int             seed ;
        int             maxneighx, maxneighy ; /* Starting area */
        int             minneighx, minneighy ; /* Final area */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;
        int             wrap_flag = 0 ;
        int             train_flag, train_discrete ;

        char            training_file[30], net_file[30], first_net_file[30] ;
        char            temp_file[15] ;
        char            net_name[15] ;

        struct curve {
                int             type ;
                double          maxgain ;
                double          mingain ;
                double          midgain ;
                int             midtime ;
                } gcurve ;

        struct flg {
                int             rnd_in ;
                } flag ;
        extern int      word_counter ;
        extern double   innput[100][100][2] ;
        int             f_xsize, f_ysize ;

init (map)
```

```
        float               map[20][20][100][2] ;
{
        int                 r, c, i ;
        float               max_rand = pow(2.0, 31.0) - 1.0 ;

        nodes = ysize * xsize ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        conscience[r][c] = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                map[r][c][i][0] = f_xsize*(rand () / max_rand) ;
                                map[r][c][i][1] = f_ysize*(rand () / max_rand) ;
                                }
                        }
                }
}

mindist (inp, close, its)
        double              inp[100][2] ;
        int                 close[2] ;
        long                its ;
{
        int                 r, c, i ;
        double              dist ;
        double              minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        if (conscience[r][c] < consc * its / nodes ) {
                                for (i = 0 ; i < number_inputs ; i++)
                                        dist+ =fabs(inp[i][0]-map[r][c][i][0])+
                                        fabs(inp[i][1]-map[r][c][i][1]);
                                if (dist < minimum) {
                                        minimum = dist ;
                                        close[0] = c ;
                                        close[1] = r ;
                                        }
                                }
                        }
                }
        conscience[close[1]][close[0]] += 1 ;
}

userinp ()
{
        int                 line ;
        int                 c ;

        do {
                printf (“TWOBAS4: Train 2nd Koh with 2-D trajectories ... \n\n”) ;
                f_xsize = (f_ysize = 15) ;

                printf (“Enter conscience factor (> 1.0): [float] ”) ;
                scanf (“%f”, &consc) ;
                if (consc < 1.0)
                        consc = 1.5 ;

                printf(“Enter size 'm n' (for an m x n) of array = ? [int int] ”) ;
                scanf(“%d %d”, &ysize, &xsize) ;
if (ysize < 2)
        ysize = 2 ;
else if (ysize > 20)
        ysize = 20 ;
if (xsize < 2)
        xsize = 2 ;
else if (xsize > 20)
        xsize = 20 ;

printf ("Do you want 0) sequential training,\n") ;
printf (" 1) randomized training? ") ;
scanf ("%d", &flag.rnd_in) ;

train_discrete = 1 ;
number_inputs = 100 ;

printf("Enter name of training file [less .dat]: ") ;
scanf ("%s", net_name) ;
sprintf (training_file, "%s.dat", net_name) ;

printf("Enter name of net file to create [less .net]: ") ;
scanf ("%s", net_name) ;
sprintf (net_file, "%s.net", net_name) ;

printf ("Number of iterations = ? [int] ") ;
scanf ("%ld", &count) ;
if (count <= 10 || count > 200000)
        count = 100 ;
mcount = (double) count ;

printf ("Number of iterations between status messages = ? [int] ") ;
scanf ("%d", &graph) ;
if (graph < 1 || graph > count)
        graph = 10 ;

ingain () ;

printf ("Starting size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &maxneighy, &maxneighx) ;
if (maxneighx < 2 || maxneighx > xsize - 1)
        maxneighx = 2 ;
if (maxneighy < 2 || maxneighy > ysize - 1)
        maxneighy = 2 ;

printf ("Final size of neighborhoods 'yn xn' = ? [int int] ") ;
scanf ("%d %d", &minneighy, &minneighx) ;
if (minneighx < 1 || minneighx > maxneighx)
        minneighx = 1 ;
if (minneighy < 1 || minneighy > maxneighy)
        minneighy = 1 ;

printf ("Initial seed for random # generator = ? [int] ");
scanf ("%d", &seed) ;
if (seed == 0) {
        seed = 138 ;
        }
srand (seed) ;

wrap_flag = 0 ;
```

```
                printf(“Ready to begin? (y/n) ”) ;
                while ((c = getc (stdin)) = = ' ' || c = = '\n' || c = = '\t')
                        ;
                } while (c != 'y') ;
}

ingain ()
{
        int             line ;

        printf(“For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR : ”);
        scanf (“%d”, &gcurve.type) ;

        if (gcurve.type = = 0 || gcurve.type = = 1) {
                printf (“Maximum gain = ? [float] ”) ;
                scanf (“%E”, &gcurve.maxgain) ;
                if (gcurve.maxgain > = 1.0 || gcurve.maxgain < = 0.0)
                        gcurve.maxgain = .99 ;

                printf (“Minimum gain = ? [float] ”) ;
                scanf (“%E”, &gcurve.mingain) ;
                if (gcurve.mingain < = 0.0 || gcurve.mingain > = 1.0)
                        gcurve.mingain = 0.0 ;
                }
        else {
                printf (“First segment starting gain = ? [float] ”) ;
                scanf (“%E”, &gcurve.maxgain) ;
                if (gcurve.maxgain > = 1.0 || gcurve.maxgain < = 0.0)
                        gcurve.maxgain = .99 ;

                printf (“Second segment starting gain = ? [float] ”) ;
                scanf (“%E”, &gcurve.midgain) ;
                if (gcurve.midgain < = 0.0 || gcurve.midgain > = 1.0)
                        gcurve.midgain = 0.0 ;

                printf (“Second segment starting iteration = ? [float] ”) ;
                scanf (“%d”, &gcurve.midtime) ;
                if (gcurve.midtime < = 0 || gcurve.midtime > count)
                        gcurve.midtime = count / 2 ;

                gcurve.mingain = 0.0 ;
                }
}

getgain (i)
        long    i ;
{
        if (gcurve.type = = 0)
                gain = (percent * (gcurve.maxgain - gcurve.mingain)) +
                        gcurve.mingain ;
        else if (gcurve.type = = 1)
                gain = 0.9 * (gcurve.maxgain - gcurve.mingain) / (1.0 + exp (i -
                        count / 2.0)) + .1 ;
        else {
                if (i < gcurve.midtime)
                        gain = gcurve.maxgain * (1.0 - (double) i / gcurve.midtime) ;
                else
                        gain = gcurve.midgain * (1.0 - (double) i / count) ;
                }
}
```

```
save_net ()
{
        int             r, c, i, x, y ;
        FILE            *fnet ;
        float           rem_x, rem_y ;

        fnet = fopen(net_file,"w") ;
        fprintf (fnet,"%d %d %d", ysize, xsize, number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                x = map[r][c][i][0] ;
                                rem_x = map[r][c][i][0] - x ;
                                y = map[r][c][i][1] ;
                                rem_y = map[r][c][i][1] - y ;
                                if (rem_x > 0.49)
                                        x++ ;
                                else if (rem_x < -0.49)
                                        x-- ;
                                if (rem_y > 0.49)
                                        y++ ;
                                else if (rem_y < -0.49)
                                        y-- ;
                                if (x < -1)
                                        x = -1 ;
                                else if (x >= f_xsize)
                                        x = f_xsize - 1 ;
                                if (y < -1)
                                        y = -1 ;
                                else if (y >= f_ysize)
                                        y = f_ysize - 1 ;
                                fprintf (fnet," %d %d", x, y) ;
                                }
                        }
                }
        fclose (fnet) ;
}

main()
{
        long            i ;
        char            s1[10] ;
        int             ws_id = 1 ;
        int             clear_flag = 1;
        FILE            *tf ;
        extern unsigned _stklen ;

        _stklen = 8192 ;
        userinp () ; /* Get input values */
        nfactorx = maxneighx - minneighx + 1 ;
        nfactory = maxneighy - minneighy + 1 ;
        init (map) ; /* Initialize weights */
        read_trn_file () ;

        for (i = 1 ; i <= count ; i++) {
                if (i % graph == 0) {
                        printf ("TWOBAS4: gain = %f, yrange = %d, ",
                                gain, nrangey) ;
                        printf ("xrange = %d, iteration # %d", nrangex,i) ;
                        printf (" (of %ld)\n", count) ;
                        if (access (net_file,0) = = 0)
                                delete (net_file) ;
                        save_net () ;
                        }
                percent = (mcount - i) / mcount ;
                getgain (i) ;
                if (flag.rnd_in = = 0)
                        getin () ;
                else
                        get_rnd_in () ;
                mindist (&innput[word_counter][0][0], closest, i) ;
                if (gcurve.type != 2) {
                        nrangex = minneighx + percent * nfactorx ;
                        nrangey = minneighy + percent * nfactory ;
                        }
                else if (i < gcurve.midtime) {
                        nrangex = minneighx + nfactorx *
                        ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        nrangey = minneighy + nfactory *
                        ((double)(gcurve.midtime - i)) / gcurve.midtime ;
                        }
                else {
                        nrangex = minneighx ;
                        nrangey = minneighy ;
                        }
                neigh[0] = nrangex ;
                neigh[1] = nrangex ;
                weightem (map) ;
                }
        save_net () ;
        printf ("\nNet file: %s saved!\n", net_file) ;
}

/*
******************************** nweight44.c ********************************

        These routines support TWOBAS4.C/EXE in training a second
        Kohonen neural net. Inputs are stored in a *.dat file.

        Inputs are 100 x-y pair trajectories filled with trailing -1's.
        ************************************************************
                G. BARMORE25 AUG 88
*/

# include math
# include stdio
# include stat

        double          innput[100][100][2] ;/* input vectors */

        extern double   input[100][2] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char             first_net_file[30] ;

        int                     number_discretes ;
        int                     word_counter ;
        int                     num_words ;
        char                    word_number[100][15] ;

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     j, i, r, c, x, y ;

        fnet = fopen (training_file, “r”) ;
        fscanf (fnet, “%d”, &num_words) ;
        for (i = 0 ; i < num_words ; i++) {
                for (j = 0 ; j < number_inputs ; j++){
                        fscanf (fnet, “%d %d”, &x, &y) ;
                        innput[i][j][0] = x ;
                        innput[i][j][1] = y ;
                        }
                }
        fclose (fnet) ;
        word_counter = -1 ;
}

getin ()
{
        int                     j ;

        word_counter++ ;
        if (word_counter == num_words)
                word_counter = 0 ;
}

get_rnd_in ()
{
        int                     i, j ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
}

weightem (map)
        float                   map[20][20][100][2] ;
{
        int                     nright, nleft, nup, ndown, r , c, i ;

        if (neigh[0] > 0 && neigh[1] > 0) {
                nright = closest[0] + neigh[0] - 1 ;
                if (nright >= xsize)
                        nright = xsize - 1 ;
                nleft = closest[0] - neigh[0] + 1 ;
                if (nleft < 0)
                        nleft = 0 ;
                nup = closest[1] - neigh[1] + 1 ;
                if (nup < 0)
                        nup = 0 ;
                ndown = closest[1] + neigh[1] - 1 ;
                if (ndown >= ysize)
```

```
            ndown = ysize - 1 ;
        }
    else {
        nright = closest[0] ;
        nleft = closest[0] ;
        nup = closest[1] ;
        ndown = closest[1] ;
        }

    for (r = nup; r <= ndown ; r++) {
        for (c = nleft ; c <= nright ; c++) {
            for (i = 0 ; i < number_inputs ; i++) {
                map[r][c][i][0] += gain *
                    (innput[word_counter][i][0] - map[r][c][i][0]) ;
                map[r][c][i][1] += gain *
                    (innput[word_counter][i][1] - map[r][c][i][1]) ;
                }
            }
        }
}
```

```
$ link twomask5,options_file/opt
/*
******************************** twomask5.c ********************************

        Routine to create *.msk file from *.net file where *.msk file
        is array of integers mask[20][20] corresponding to nodes of *.net
        file.

        Each integer is the number of weights which are not -1.
        Trajectories (node weights) are 100 x-y pairs filled with
        trailing -1's.
        *************************************************************
*/
# include math
# include stdio

        int             mask[20][20] ;
        float           map[20][20][100][2] ; /* output nodes */
        double          node_dist ;

        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

non_zero (map)
        float           map[20][20][100][2] ;
{
        int             r, c, i, number, x, y ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = number_inputs+1 ;
                        y = (x = -1) ;
                        while ((y = = -1) && (x = = -1) && (number > 1)){
                                number-- ;
                                x = map[r][c][number-1][0] ;
                                y = map[r][c][number-1][1] ;
                                printf ("%2d %2d ", x, y) ;
                                }
                        printf ("\n**** %d ****\n", number) ;
                        mask[r][c] = number ;
                }
        }
}

main ()
{
        printf ("\nTWOMASK4 (Creates net mask for 2-D trajectories\n\n) ... ") ;
        find_mask () ;
}

find_mask ()
{
        int             r, c, i, x, y ;
        FILE            *fnet ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, “%s.net”, temp_file) ;
        sprintf (training_file, “%s.mak”, temp_file) ;

        fnet = fopen (net_file, “r”) ;
        fscanf (fnet,”%d %d %d”, &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet,” %d %d”, &x, &y) ;
                                map[r][c][i][0] = x ;
                                map[r][c][i][1] = y ;
                                }
                        }
                }
        fclose (fnet) ;
        non_zero (map) ;
        save_mask () ;
}

save_mask ()
{
        FILE                    *fmask ;
        int                     r, c ;

        fmask = fopen (training_file, “w”) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fprintf (fmask, “%d ”, mask[r][c]) ;
                        }
                }
        fclose (fmask) ;
}
```

```
$ link twopic4c,nplot,nprinter,mat3,nwt4,options_file/opt
/*
******************************** twopic4c.c ********************************
            This routine is used to show graphically those nodes which 'light
            up' using Euclidean distance when the training set of inputs is
            applied to a second Kohonen net.

            Inputs are 100 x-y pair trajectories filled with trailing -1's.
            ************************************************************
*/
# include math
# include stdio
# include time

            float           map[20][20][100][2] ; /* output nodes */
            double          input[100][2] ; /* input nodes */
            double          gain, noise ;
            double          mcount ;
            double          percent ;
            double          xoff = 0.0 ;
            double          yoff = 0.0 ;
            double          node_dist ;

            int             closest[2] ; /* closest node */
            int             neigh[2] ; /* neighbor */
            int             nrangex, nrangey ; /* neighbor range */
            int             nfactorx, nfactory ; /* neighbor factor */
            long            count ; /* # of iterations */
            int             graph ; /* # between plots */
            int             seed ;
            int             maxneighx, maxneighy ; /* Starting area */
            int             minneighx, minneighy ; /* Final area */
            int             xsize, ysize ; /* Size of array */
            int             number_inputs ;
            int             wrap_flag = 0 ;
            int             train_flag, train_discrete ;

            char            training_file[30], temp_file[30], first_net_file[30] ;
            char            net_file[30] ;

            struct curve {
                        int             type ;
                        double          maxgain ;
                        double          mingain ;
                        double          midgain ;
                        int             midtime ;
                        } gcurve ;

            extern int      xy[] ; /* array holding x,y */
            extern double   xdel, ydel ;
            extern double   xlow, xup, ylow, yup ;
            extern int      num_words ;
            extern char     word_number[100][15] ;

mindist (map, inp, close)
            double          inp[100][2] ;
            int             close[2] ;
            float           map[20][20][100][2] ;
{
            int             r, c, i ;
```

```
        double  dist ;
        double  minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++)
                                dist += fabs(inp[i][0] - map[r][c][i][0]) +
                                        fabs(inp[i][1] - map[r][c][i][1]) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int     c ;

        printf ("\nTWOPIC4c (Plot Words for 2-D Reduced Queued Traj)...\n") ;
        map_speech () ;
}

map_speech ()
{
        int     r, c, i, j, k, x, y ;
        char    sub_title[60], temp[30] ;
        char    name_trj[20] ;
        int     loc[125][2] ;
        FILE    *fnet, *flog ;
        int     sound ;
        short   length ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;

        read_trn_file () ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %d %d", &x, &y) ;
                                map[r][c][i][0] = x ;
                                map[r][c][i][1] = y ;
                                }
                        }
                }
        fclose (fnet) ;

        sprintf (sub_title, "%s -> %s ->", first_net_file, net_file) ;
        sprintf (name_trj, "%s.trj", temp) ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "TWOPIC4c: %s", name_trj) ;

        graph_test (training_file) ;
        length = (short) strlen (sub_title) ;
        draw_grid2 (ysize, xsize, sub_title, length) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        for (sound = 0 ; sound < num_words ; sound++) {
                getin () ;
                mindist (map, input, &loc[sound][0]) ;
                printf ("%d : [%d,%d] dist = %le\n",
                        sound, loc[sound][0], loc[sound][1], node_dist) ;
                fprintf (flog, "%d : [%d,%d] dist = %le\n",
                        sound, loc[sound][0], loc[sound][1], node_dist) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
        draw_speech_map (sound, loc) ;
        scanf ("%s",temp) ;
        clipoff () ;
        graphoff () ;
}

graph_test (name)
        char    name[30] ;
{
        char    title[79], labelx[79] ;
        float   xloc[5] = {0, 639.0, 639.0, 0.0, 0.0} ;
        float   yloc[5] = {349.0, 349.0, 0.0, 0.0, 349.0} ;
        int     points = 5 ;
        int     ws_id = 1 ;
        int     clear_flag = 1 ;
        short   length ;

        sprintf(title,"TWOPIC4c: Kohonen TWO NETS -- %s", name) ;
        sprintf(labelx," ") ;
        graphon () ;
        gks$clear_ws (&ws_id, &clear_flag) ;
        gks$polyline (&points, xloc, yloc) ;
        prepcolmat (ysize, xsize) ;
        length = (short) strlen (title) ;
        outtitle (title, length) ;
        length = (short) strlen (labelx) ;
        outlabelx (labelx, length) ;
        clipon () ;
}

/*
******************************** nwt4.c ********************************

        These routines support TWOPIC4C.C/EXE in finding which node lights
        up in a second Kohonen net when a training digit is applied.
```

```
          Inputs are 100 x-y pair trajectories filled with trailing -1's.
          Euclidean distance is used in finding which node lights up.
          ******************************************************************
*/

# include math
# include stdio
# include stat

          extern double       input[100][2] ; /* input nodes */
          extern double       gain ;

          extern int          closest[2] ; /* closest node */
          extern int          neigh[2] ; /* neighbor */
          extern int          xsize, ysize ; /* Size of array */
          extern int          number_inputs ;
          extern int          train_discrete ;
          extern char         training_file[30] ;
          extern char         first_net_file[30] ;

          int                 number_discretes ;
          int                 word_counter ;
          int                 num_words ;
          char                word_number[100][15] ;

          int                 f_ysize, f_xsize, f_number_inputs ;
          float               f_map[20][20][16] ;

read_trn_file ()
{
          FILE                *tf, *fnet ;
          int                 i, r, c ;

          tf = fopen (training_file, "r") ;
          fscanf (tf, "%d", &num_words) ;
          for (i = 0 ; i < num_words ; i++)
                    fscanf (tf, "%s", word_number[i]) ;
          fclose (tf) ;
          word_counter = 0 ;

          fnet = fopen (first_net_file, "r") ;
          fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
          for (r = 0 ; r < f_ysize ; r++) {
                    for (c = 0 ; c < f_xsize ; c++) {
                              for (i = 0 ; i < f_number_inputs ; i++) {
                                        fscanf (fnet," %f", &f_map[r][c][i]) ;
                                        }
                              }
                    }
          fclose (fnet) ;
}

getin ()
{
          if (word_counter == num_words)
                    word_counter = 0 ;
          read_word (word_counter) ;
          word_counter++ ;
}
```

```
get_rnd_in ()
{
        int                     i ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

read_word (pointer)
        int                     pointer ;
{
        int                     flag, r, c, i, j, k ;
        double                  in[16], d1, d2, d3, d4, d5 ;
        float                   element ;

        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int                     sound, point, x[5], y[5] ;
        int                     max_pts ;
        double                  max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        if (j >= number_inputs)
                j = number_inputs ;
        for (i = 0 ; i < j ; i++) {
                input[i][0] = loc2[i][0] ;
                input[i][1] = loc2[i][1] ;
                }
        for (i = j ; i < number_inputs ; i++) {
```

```
                        input[i][0] = (input[i][1] = -1.0) ;
                        }
}

f_mindist (f_map, inp, close)
            double                  inp[16] ;
            int                     close[2] ;
            float                   f_map[20][20][16] ;
{
            int                     r, c, i ;
            double                  dist ;
            double                  minimum = 9.9e31 ;

            for (r = 0 ; r < f_ysize ; r++) {
                        for (c = 0 ; c < f_xsize ; c++) {
                                    dist = 0.0 ;
                                    for (i = 0 ; i < f_number_inputs ; i++)
                                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                                    if (dist < minimum) {
                                                minimum = dist ;
                                                close[0] = c ;
                                                close[1] = r ;
                                                }
                                    }
                        }
}
```

```
$ link twopic8b,nwin8b,options_file/opt
/*
********************************* twopic8b.c *********************************

          This routine finds the closest digit (from a group of digits) to
          each node in a second Kohonen net. In particular, this routine
          uses 100 x-y pair trajectories filled with trailing -1's.

          For each node, the routine searches through the whole list for the
          digit 'closest' to that node's weight. 'Closest' in this case
          is found through a mini-DTW that uses both the node's masked
          length (without trailing -1's) and the length of the trajectory
          (also without -1's).

          Data output includes the node number, distance to the digit
          found closest, and the name of the *.trn file of the digit.
          ***************************************************************
*/
# include math
# include stdio
# include time

          float           map[20][20][100][2] ; /* output nodes */
          double          input[100][2] ; /* input nodes */
          double          node_dist ;

          int             closest[2] ; /* closest node */
          int             xsize, ysize ; /* Size of array */
          int             number_inputs ;

          char            training_file[30], temp_file[30], first_net_file[30] ;
          char            net_file[30] ;

          extern int      num_words ;
          int             mask[20][20] ;
          int             map2[20][20][100][2] ;
          float           aa = 0.75 ;
          float           bb = 0.75 ;
          extern int      f_xsize, f_ysize ;

          extern int      location[2000][2] ;
          int             innput[200][100][2] ;
          extern int      length[200] ;
          extern char     word_number[200][15] ;

mindist (r, c, close)
          int             r, c ;
          int             *close ;
{
          int             sound ;
          double          dist ;
          double          minimum = 9.99e31 ;

          for (sound = 0 ; sound < num_words ; sound++) {
                  dtw (&map2[r][c][0][0], &innput[sound][0][0], mask[r][c],
                          length[sound], &dist) ;
                  if (dist < minimum) {
                          minimum = dist ;
                          *close = sound ;
                          }
```

```
                }
        node_dist = minimum ;
}

main ()
{
        int             c ;

        printf ("\nTWOPIC8b (Closest word for each node: 100 wts/2-D )...\n") ;
        map_speech () ;
}

map_speech ()
{
        int             r, c, i, j, k, x, y ;
        char            name_trj[30], temp[30] ;
        int             loc ;
        FILE            *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;
        sprintf (temp, "%s.msk", temp_file) ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %d %d", &x, &y) ;
                                map2[r][c][i][0] = x ;
                                map2[r][c][i][1] = y ;
                                }
                        }
                }
        fclose (fnet) ;

        read_trn_file () ;

        fmask = fopen (temp, "r") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fscanf (fmask,"%d", &mask[r][c]) ;
                        }
                }
        fclose (fmask) ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "TWOPIC8b:\n") ;
        fprintf (flog, "-> %s -> %s ->\n", first_net_file, net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;
        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        printf ("\nReading word: ") ;
        for (r = 0 ; r < num_words ; r++) {
                printf ("%d ", r) ;
                getin () ;
                for (c = 0 ; c < number_inputs ; c++) {
                        innput[r][c][0] = location[c][0] ;
                        innput[r][c][1] = location[c][1] ;
                        }
                }
        printf ("\n") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        mindist (r, c, &loc) ;
                        k = c + r * xsize ;
                        printf ("\nNode %3d : word # %3d, dist = %le ",
                                k, loc, node_dist) ;
                        printf ("(%s)", word_number[loc]) ;
                        fprintf (flog, "\nNode %3d : word # %3d, dist = %le ",
                                k, loc, node_dist) ;
                        fprintf (flog, "(%s)", word_number[loc]) ;
                        }
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int     template[200][2], utterance[200][2] ;
        int     t_length, u_length ;
        double  *ave_dist ;
{
        float   back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;

        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        if (b_p[ptr][t_length-1] != 0)
                *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
        else
                *ave_dist = 9.9e31 ;
}

/*
******************************* nwin6b.c ******************************

        These routines support TWOPIC8B.C/EXE in finding a digit from a
        specified set that is closest to each node in a second Kohonen net.

        Inputs are 100 x-y pair trajectories filled with trailing -1's.
        ***********************************************************
                                G. Barmore25 Aug 88
*/

# include math
# include stdio
# include stat

        extern double   input[100][2] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char                    word_number[200][15] ;
        int                     length[200] ;

        int                     f_ysize, f_xsize, f_number_inputs ;
        float                   f_map[20][20][16] ;
        int                     location[2000][2] ;
        extern int              map2[20][20][100][2] ;
        extern float            map[20][20][100][2] ;

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     i, r, c, k, temp ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;
        word_counter = 0 ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

get_rnd_in ()
{
        int                     i ;
        double                  max_rand = pow (2.0, 31.0) - 1.0 ;
        int                     pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

read_word (pointer)
        int                     pointer ;
{
        int                     flag, r, c, i, j, k ;
        double                  in[16], d1, d2, d3, d4, d5 ;
        float                   element ;

        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[pointer] = j ;
        for (i = j ; i < 2000 ; i++)
                location[i][1] = (location[i][0] = -1) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
```

```
                    }
                }
            }
    }
```

```
$ link twopic6b,nwin5b,lookup6,options_file/opt
/*
******************************** twopic6.c ********************************

        This routine finds the closest node in a second Kohonen net (and
        the digit assigned to it from twopic8b.c) to each digit in a set
        of test digits.digits). In particular, this routine
        uses 100 x-y pair trajectories filled with trailing -1's.

        For each test digit, the routine searches through every node in the
        Kohonen net for the node 'closest' to that digit. 'Closest' in this
        case is found through a mini-DTW that uses both the node's masked
        length (without trailing -1's) and the length of the trajectory
        (also without -1's).

        Once the closest node is identified, the digit assigned to it
        from twopic8b is found by a look-up table (lookup6.c).

        Data output includes the digit number (actually the place in the
        test set that the digit resides at), distance to the node
        found closest, and the name of the *.trn file of the digit, and
        the digit assigned to the closest node.

        Thus, this is the end test of the second Kohonen net. Will
        test digits light up a node assigned to the same class of digit?
        *****************************************************************
*/
# include math
# include stdio
# include time

        float           map[20][20][100][2] ; /* output nodes */
        double          input[100][2] ; /* input nodes */
        double          node_dist ;

        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        extern int      num_words ;
        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        extern int      f_xsize, f_ysize ;
        extern int      length[200] ;
        extern int      location[2000][2] ;

mindist (close)
        int             close[2] ;
{
        int             r, c, i ;
        double          dist ;
        double          distance ;
        double          minimum = 9.99e31 ;
        double          p1, p2 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dtw (&map2[r][c][0][0], location, mask[r][c],
                                length[0], &dist) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int     c ;

        printf ("\nTWOPIC6b (DTW Words for 2-D Reduced Queued Traj)...\n") ;
        map_speech () ;
}

map_speech ()
{
        int     r, c, i, j, k, x, y ;
        char    sub_title[60], temp[30] ;
        char    name_trj[20] ;
        int     loc[2] ;
        FILE    *fnet, *flog, *fmask ;
        int     sound ;
        short   length ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;
        sprintf (temp, "%s.msk", temp_file) ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %d %d", &x, &y) ;
                                map2[r][c][i][0] = x ;
                                map2[r][c][i][1] = y ;
                                }
                        }
                }
        fclose (fnet) ;

        read_trn_file () ;

        fmask = fopen (temp, "r") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fscanf (fmask,"%d", &mask[r][c]) ;
                        }
                }
        fclose (fmask) ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "TWOPIC6b:\n") ;
        fprintf (flog, "-> %s -> %s ->\n", first_net_file, net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (sound = 0 ; sound < num_words ; sound++) {
                getin () ;
                mindist (loc) ;
                k = loc[0] + loc[1] * xsize ;
                printf ("\n%d : [%d,%d] dist = %le ",
                        sound, loc[0], loc[1], node_dist) ;
                print_digit (k) ;
                fprintf (flog, "\n%d : [%d,%d] dist = %le ",
                        sound, loc[0], loc[1], node_dist) ;
                fprint_digit (k, flog) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int     template[200][2], utterance[200][2] ;
        int     t_length, u_length ;
        double  *ave_dist ;
{
        float   back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;

        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

/*
********************************* nwin5b.c *********************************

        These routines support TWOPIC6*.C/EXE in finding the closest node
        to a given input digit in a second Kohonen net.

        Trajectories are 100 x-y pairs filled with trailing -1's.
        *****************************************************************
*/

# include math
# include stdio
# include stat

        extern double   input[100][2] ; /* input nodes */
        extern double   gain ;

        extern int      closest[2] ; /* closest node */
        extern int      neigh[2] ; /* neighbor */
        extern int      xsize, ysize ; /* Size of array */
        extern int      number_inputs ;
        extern int      train_discrete ;
        extern char     training_file[30] ;
        extern char     first_net_file[30] ;

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[200][15] ;
        int             length[200] ;
        int                 f_ysize, f_xsize, f_number_inputs ;
        float               f_map[20][20][16] ;
        int                 location[2000][2] ;
        extern int          map2[20][20][100][2] ;
        extern float        map[20][20][100][2] ;

read_trn_file ()
{
        FILE                *tf, *fnet ;
        int                 i, r, c, k, temp ;

        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;
        word_counter = 0 ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

get_rnd_in ()
{
        int                 i ;
        double              max_rand = pow (2.0, 31.0) - 1.0 ;
        int                 pointer ;

        pointer = floor ((rand() * (num_words - .0001) / max_rand)) ;
        read_word (pointer) ;
}

read_word (pointer)
        int                 pointer ;
{
        int                 flag, r, c, i, j, k ;
        double              in[16], d1, d2, d3, d4, d5 ;
        float               element ;

        int                 loc2[2000][2], loc3[2000][2] ;
        FILE                *fsnd ;
        int                 sound, point, x[5], y[5] ;
        int                 max_pts ;
        double              max ;
        fand = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fand, "%f", &element) ;
                if (feof(fand) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fand) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 2000 ; i++)
                location[i][1] = (location[i][0] = -1) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 99999.9 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
```

```
$ link twopic8c,nwin6b,options_file/opt
/*
******************************** twopic8c.c ********************************

            This routine finds the closest digit (from a group of digits) to
            each node in a second Kohonen net. In particular, this routine
            uses 100 x-y pair trajectories filled with trailing -1's.

            For each node, the routine searches through the whole list for the
            digit 'closest' to that node's weight. 'Closest' in this case
            is found through a TAXI distance.

            Data output includes the node number, distance to the digit
            found closest, and the name of the *.trn file of the digit.
            ************************************************************
*/
# include math
# include stdio
# include time

            float             map[20][20][100][2] ; /* output nodes */
            double            input[100][2] ; /* input nodes */
            double            node_dist ;

            int               closest[2] ; /* closest node */
            int               xsize, ysize ; /* Size of array */
            int               number_inputs ;

            char              training_file[30], temp_file[30], first_net_file[30] ;
            char              net_file[30] ;

            extern int        num_words ;
            int               mask[20][20] ;
            int               map2[20][20][100][2] ;
            float             aa = 0.75 ;
            float             bb = 0.75 ;
            extern int        f_xsize, f_ysize ;

            extern int        location[2000][2] ;
            int               innput[200][100][2] ;
            extern int        length[200] ;
            extern char       word_number[200][15] ;

mindist (r, c, close)
            int               r, c ;
            int               *close ;
{
            int               i, sound ;
            int               comp_length ;
            double            dist ;
            double            minimum = 9.99e31 ;

            for (sound = 0 ; sound < num_words ; sound++) {
                        if (length[sound] > mask[r][c])
                                    comp_length = length[sound] ;
                        else
                                    comp_length = mask[r][c] ;
                        dist = 0.0 ;
                        for (i = 0 ; i < comp_length ; i++) {
                                    dist += abs (map2[r][c][i][0] - innput[sound][i][0]) +
                                abs (map2[r][c][i][1] - innput[sound][i][1]) ;
                        }
                if (dist < minimum) {
                        minimum = dist ;
                        *close = sound ;
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int     c ;

        printf ("\nTWOPIC8c (Closest (taxi) to each node: 100 wts/2-D )...\n") ;
        map_speech () ;
}

map_speech ()
{
        int     r, c, i, j, k, x, y ;
        char    name_trj[30], temp[30] ;
        int     loc ;
        FILE    *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter name of output Koh net_file [less .net]: ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.net", temp_file) ;
        sprintf (temp, "%s.msk", temp_file) ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet,"%d %d %d", &ysize, &xsize, &number_inputs) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %d %d", &x, &y) ;
                                map2[r][c][i][0] = x ;
                                map2[r][c][i][1] = y ;
                                }
                        }
                }
        fclose (fnet) ;

        read_trn_file () ;

        fmask = fopen (temp, "r") ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        fscanf (fmask,"%d", &mask[r][c]) ;
                        }
                }
        fclose (fmask) ;
        flog = fopen (“temp.log”,“w”) ;
        fprintf (flog, “TWOPIC8b:\n”) ;
        fprintf (flog, “-> %s -> %s ->\n”, first_net_file, net_file) ;
        fprintf (flog, “Size is %d by %d nodes\n”, xsize, ysize) ;

        printf (“\nExpect %d calculations.\n”, num_words) ;
        fprintf (flog, “Expect %d calculations.\n”, num_words) ;
        printf (“\nReading word: ”) ;
        for (r = 0 ; r < num_words ; r++) {
                printf (“%d ”, r) ;
                getin () ;
                for (c = 0 ; c < number_inputs ; c++) {
                        innput[r][c][0] = location[c][0] ;
                        innput[r][c][1] = location[c][1] ;
                        }
                }
        printf (“\n”) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        mindist (r, c, &loc) ;
                        k = c + r * xsize ;
                        printf (“\nNode %3d : word # %3d, dist = %le ”,
                                k, loc, node_dist) ;
                        printf (“(%s)”, word_number[loc]) ;
                        fprintf (flog, “\nNode %3d : word # %3d, dist = %le ”,
                                k, loc, node_dist) ;
                        fprintf (flog, “(%s)”, word_number[loc]) ;
                        }
                }
        printf (“\nCalculations finished.\n”) ;
        fclose (flog) ;
}
```

```
$ link codebk,options_file/opt
/*
******************************** codebk.c ********************************

        Routine to generate and test an untrained second Kohonen net.

        An untrained net is generate by taking stored trajectories from
        path.dat and storing them as weights in respective nodes. That
        is, the first row are assigned the zeros through to the last
        row assigned the nines.

        A test set of digits is specified, and tested using DTW(mask[],
        length[]). All trajectories are 100scalars filled with
        trailing 0's.
        ****************************************************************
                                G. BARMORE1 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dtw (&map2[r][c][0][0], location, mask[r][c],
                                length[0], &dist) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                *close = c ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int             c ;

        printf ("\nCODEBK: Uses path.dat as codebook...\n") ;
        map_speech () ;
}

map_speech ()
{
        int             r, c, i, j, k ;
        char            name_trj[30], temp[30] ;
        int             loc ;
        FILE            *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        read_trn_file () ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "CODEBK: %s\n", name_trj) ;
        fprintf (flog, "-> %s -> path.dat ->\n", first_net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (r = 0 ; r < num_words ; r++) {
                getin () ;
                mindist (&loc) ;
                printf ("Word %3d is: %3d\n", r, loc) ;
                fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int             template[200][2], utterance[200][2] ;
        int             t_length, u_length ;
        double          *ave_dist ;
{
        float           back_path[2][200] ;
        int             b_p[2][200] ;
        int             r, c ;
        int             ptr, b_ptr ;
        float           d1, d2, d3, dist ;
        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     j, i, r, c ;
        int                     temp, k, number ;


        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
```

```
                        fscanf (tf, “%s”, word_number[i]) ;
            fclose (tf) ;

            fnet = fopen (first_net_file, “r”) ;
            fscanf (fnet,“%d %d %d”, &f_ysize, &f_xsize, &f_number_inputs) ;
            for (r = 0 ; r < f_ysize ; r++) {
                        for (c = 0 ; c < f_xsize ; c++) {
                                    for (i = 0 ; i < f_number_inputs ; i++) {
                                                fscanf (fnet,“ %f”, &f_map[r][c][i]) ;
                                                }
                                    }
                        }
            fclose (fnet) ;

            ysize = (xsize = 10) ;
            number_inputs = 100 ;

            fnet = fopen (“path.dat”, “r”) ;
            fscanf (fnet, “%d”, &num_words) ;
            for (r = 0 ; r < ysize ; r++) {
                        for (c = 0 ; c < xsize ; c++) {
                                    for (i = 0 ; i < number_inputs ; i++) {
                                                fscanf (fnet,“ %le”, &map[r][c][i]) ;
                                                }
                                    }
                        }
            fclose (fnet) ;

            for (r = 0 ; r < ysize ; r++) {
                        for (c = 0 ; c < xsize ; c++) {
                                    number = 0 ;
                                    for (i = 0 ; i < number_inputs ; i++) {
                                                if (map[r][c][i] > 5.0e-4)
                                                            number++ ;
                                                temp = (int) (map[r][c][i] * 226.0 - 1.0) ;
                                                k = temp % f_xsize ;
                                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                                map2[r][c][i][0] = k ;
                                                }
                                    mask[r][c] = number ;
                                    }
                        }
            word_counter = 0 ;
}

getin ()
{
            if (word_counter == num_words)
                        word_counter = 0 ;
            read_word (word_counter) ;
            word_counter++ ;
}

read_word (pointer)
            int         pointer ;
{
            int         flag, r, c, i, j, k ;
            double      in[16], d1, d2, d3, d4, d5 ;
            float       element ;
```

```
        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
```

```
            if (dist < minimum) {
                minimum = dist ;
                close[0] = c ;
                close[1] = r ;
                }
            }
        }
}
```

```
$ link codebkb,options_file/opt
/*
******************************** codebkb.c ********************************

        Routine to generate and test an untrained second Kohonen net.

        An untrained net is generate by taking stored trajectories from
        pathold.dat and storing them as weights in respective nodes. That
        is, the first row are assigned the zeros through to the last
        row assigned the nines.

        A test set of digits is specified, and tested using DTW(mask[],
        length[]). All trajectories are 75 scalars filled with
        trailing 0's.
        ************************************************************
                                        G. BARMORE1 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dtw (&map2[r][c][0][0], location, mask[r][c],
                                length[0], &dist) ;
                        if (dist < minimum) {
```

```
                                        minimum = dist ;
                                        *close = c ;
                                        }
                                }
                        }
        node_dist = minimum ;
}

main ()
{
        int     c ;

        printf ("\nCODEBKb: Uses pathold.dat as codebook...\n") ;
        map_speech () ;
}

map_speech ()
{
        int     r, c, i, j, k ;
        char    name_trj[30], temp[30] ;
        int     loc ;
        FILE    *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        read_trn_file () ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "CODEBKb: %s\n", name_trj) ;
        fprintf (flog, "-> %s -> path.dat ->\n", first_net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (r = 0 ; r < num_words ; r++) {
                getin () ;
                mindist (&loc) ;
                printf ("Word %3d is: %3d\n", r, loc) ;
                fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int     template[200][2], utterance[200][2] ;
        int     t_length, u_length ;
        double  *ave_dist ;
{
        float   back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;
        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

read_trn_file ()
{
        FILE                    *tf, *fnet ;
        int                     j, i, r, c ;
        int                     temp, k, number ;


        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
```

```
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        ysize = (xsize = 10) ;
        number_inputs = 75 ;

        fnet = fopen ("pathold.dat", "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %le", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                if (map[r][c][i] > 5.0e-4)
                                        number++ ;
                                temp = (int) (map[r][c][i] * 225.0) ;
                                k = temp % f_xsize ;
                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                map2[r][c][i][0] = k ;
                                }
                        mask[r][c] = number ;
                        }
                }
        word_counter = 0 ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ,
        read_word (word_counter) ;
        word_counter++ ;
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;
        int                     loc2[2000][2], loc3[2000][2] ;
        FILE                    *fsnd ;
        int                     sound, point, x[5], y[5] ;
        int                     max_pts ;
        double                  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}

f_mindist (f_map, inp, close)
        double                  inp[16] ;
        int                     close[2] ;
        float                   f_map[20][20][16] ;
{
        int                     r, c, i ;
        double                  dist ;
        double                  minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
```

```
            if (dist < minimum) {
                minimum = dist ;
                close[0] = c ;
                close[1] = r ;
                }
            }
        }
}
```

```
$ link codebk2,options_file/opt
/*
******************************** codebk2.c ********************************

        Routine to generate and test an untrained second Kohonen net.

        An untrained net is generate by taking stored trajectories from
        path.dat and storing them as weights in respective nodes. That
        is, the first row are assigned the zeros through to the last
        row assigned the nines.

        A test set of digits is specified, and tested using TAXI distance.
        All trajectories are 100 scalars filled with trailing 0's.
        ************************************************************
                                                G. BARMORE1 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             i, r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                dist += abs(map2[r][c][i][0]-location[i][0])+
                                        abs(map2[r][c][i][1]-location[i][1]) ;
                                }
                        if (dist < minimum) {
                                minimum = dist ;
                                *close = c ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int             c ;

        printf ("\nCODEBK: Uses path.dat as codebook...\n") ;
        map_speech () ;
}

map_speech ()
{
        int             r, c, i, j, k ;
        char            name_trj[30], temp[30] ;
        int             loc ;
        FILE            *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        read_trn_file () ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "CODEBK: %s\n", name_trj) ;
        fprintf (flog, "-> %s -> path.dat ->\n", first_net_file) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (r = 0 ; r < num_words ; r++) {
                getin () ;
                mindist (&loc) ;
                printf ("Word %3d is: %3d\n", r, loc) ;
                fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

read_trn_file ()
{
        FILE            *tf, *fnet ;
        int             j, i, r, c ;
        int             temp, k, number ;


        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        ysize = (xsize = 10) ;
        number_inputs = 100 ;

        fnet = fopen ("path.dat", "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %le", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                if (map[r][c][i] > 5.0e-4)
                                        number++ ;
                                temp = (int) (map[r][c][i] * 226.0 - 1.0) ;
                                k = temp % f_xsize ;
                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                map2[r][c][i][0] = k ;
                                }
                        mask[r][c] = number ;
                        }
                }
        word_counter = 0 ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float           element ;

        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], “r”) ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, “%f”, &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                close[0] = c ;
                                close[1] = r ;
                                }
                        }
                }
}
```

```
$ link codebk2b,options_file/opt
/*
******************************** codebk2b.c ********************************

        Routine to generate and test an untrained second Kohonen net.

        An untrained net is generate by taking stored trajectories from
        pathold.dat and storing them as weights in respective nodes. That
        is, the first row are assigned the zeros through to the last
        row assigned the nines.

        A test set of digits is specified, and tested using TAXI distance.
        All trajectories are 75 scalars filled with trailing 0's.
        **************************************************************
                                G. BARMORE1 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             i, r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                dist+=abs(map2[r][c][i][0]-location[i][0])+
                                abs(map2[r][c][i][1]-location[i][1]) ;
```

```
                                        }
                              if (dist < minimum) {
                                        minimum = dist ;
                                        *close = c ;
                                        }
                              }
                    }
          node_dist = minimum ;
}

main ()
{
          int       c ;

          printf ("\nCODEBK2b: Uses pathold.dat & TAXI ...\n") ;
          map_speech () ;
}

map_speech ()
{
          int       r, c, i, j, k ;
          char      name_trj[30], temp[30] ;
          int       loc ;
          FILE      *fnet, *flog, *fmask ;

          printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
          scanf ("%s", temp) ;
          sprintf (first_net_file, "%s.net", temp) ;

          printf ("Enter name of header file containing words (less .hdr): ") ;
          scanf ("%s", temp_file) ;
          sprintf (training_file, "%s.hdr", temp_file) ;

          read_trn_file () ;

          flog = fopen ("temp.log","w") ;
          fprintf (flog, "CODEBKb: %s\n", name_trj) ;
          fprintf (flog, "-> %s -> path.dat ->\n", first_net_file) ;
          fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

          printf ("\nExpect %d calculations.\n", num_words) ;
          fprintf (flog, "Expect %d calculations.\n", num_words) ;
          for (r = 0 ; r < num_words ; r++) {
                    getin () ;
                    mindist (&loc) ;
                    printf ("Word %3d is: %3d\n", r, loc) ;
                    fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                    }
          printf ("\nCalculations finished.\n") ;
          fclose (flog) ;
}

read_trn_file ()
{
          FILE      *tf, *fnet ;
          int       j, i, r, c ;
          int       temp, k, number ;


          tf = fopen (training_file, "r") ;
```

```
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        ysize = (xsize = 10) ;
        number_inputs = 75 ;

        fnet = fopen ("pathold.dat", "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < number_inputs ; i++) {
                                fscanf (fnet," %le", &map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                if (map[r][c][i] > 5.0e-4)
                                        number++ ;
                                temp = (int) (map[r][c][i] * 225.0) ;
                                k = temp % f_xsize ;
                                map2[r][c][i][1] = (temp - k) / f_xsize ;
                                map2[r][c][i][0] = k ;
                                }
                        mask[r][c] = number ;
                        }
                }
        word_counter = 0 ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;

        int     loc2[2000][2], loc3[2000][2] ;
        FILE    *fsnd ;
        int     sound, point, x[5], y[5] ;
        int     max_pts ;
        double  max ;

        for (i = 0 ; i < number_inputs ; i++)
                input[i] = 0.0 ;
        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = 0) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
```

```
                                for (i = 0 ; i < f_number_inputs ; i++)
                                        dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                                if (dist < minimum) {
                                        minimum = dist ;
                                        close[0] = c ;
                                        close[1] = r ;
                                        }
                                }
                        }
}
```

```
$ link coder,options_file/opt
/*
********************************* coder.c ********************************

        This routine creates an untrained second Kohonen with stored
        trajectories from a *.dat file and test them on a specified set
        of digits using DTW distance.

        Trajectories are 100 x-y pairs filled with trailing -1's. The
        user may choose the length used in DTW distance.
        ***************************************************************
                                        G. BARMORE25 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dtw (&map2[r][c][0][0], location, mask[r][c],
                                length[0], &dist) ;
                        if (dist < minimum) {
                                minimum = dist ;
                                *close = c ;
                                }
                        }
                }
        node_dist = minimum ;
}

main ()
{
        int          c ;

        printf ("\nCODER: Create codebook using x-y pairs/dtw...\n") ;
        map_speech () ;
}

map_speech ()
{
        int          r, c, i, j, k ;
        char         name_trj[30], temp[30] ;
        int          loc ;
        FILE         *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter .dat file for net generation (less .dat): ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.dat", temp_file) ;

        printf ("Enter [number_inputs] desired (< =100): ") ;
        scanf ("%d", &number_inputs) ;

        read_trn_file () ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "CODER:\n") ;
        fprintf (flog, "%s -> %s -> %s\n", training_file,
                first_net_file, net_file ) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (r = 0 ; r < num_words ; r++) {
                getin () ;
                mindist (&loc) ;
                printf ("Word %3d is: %3d\n", r, loc) ;
                fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

dtw (template, utterance, t_length, u_length, ave_dist)
        int          template[200][2], utterance[200][2] ;
        int          t_length, u_length ;
        double       *ave_dist ;
{
        float        back_path[2][200] ;
        int     b_p[2][200] ;
        int     r, c ;
        int     ptr, b_ptr ;
        float   d1, d2, d3, dist ;

        dist = 0.0 ;
        b_ptr = 1 ;
        b_p[0][0] = 1 ;
        for (r = 1 ; r < t_length ; r++)
                b_p[0][r] = b_p[0][r-1] + 1 ;
        for (r = 0 ; r < t_length ; r++) {
                back_path[0][r] = (dist += bb * (
                        abs(utterance[0][0] - template[r][0]) +
                        abs(utterance[0][1] - template[r][1]))) ;
                }
        for (c = 1 ; c < u_length ; c++) {
                if (b_ptr ==0) {
                        b_ptr = 1 ;
                        ptr = 0 ;
                        }
                else {
                        b_ptr = 0 ;
                        ptr = 1 ;
                        }
                for (r = 0 ; r < t_length ; r++) {
                        dist = abs(utterance[c][0] - template[r][0]) +
                                abs(utterance[c][1] - template[r][1]) ;
                        if (r == 0){
                                back_path[ptr][r] = back_path[b_ptr][r] +
                                        (aa * dist) ;
                                b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                }
                        else {
                                d1 = back_path[b_ptr][r-1] + dist ;
                                d2 = back_path[ptr][r-1] + (bb * dist) ;
                                d3 = back_path[b_ptr][r] + (aa * dist) ;
                                if (d2 <= d3 && d2 < d1){
                                        back_path[ptr][r] = d2 ;
                                        b_p[ptr][r] = b_p[ptr][r-1] + 1 ;
                                        }
                                else if (d3 <= d2 && d3 < d1) {
                                        back_path[ptr][r] = d3 ;
                                        b_p[ptr][r] = b_p[b_ptr][r] + 1 ;
                                        }
                                else {
                                        back_path[ptr][r] = d1 ;
                                        b_p[ptr][r] = b_p[b_ptr][r-1] + 1 ;
                                        }
                                }
                        }
                }
        *ave_dist = back_path[ptr][t_length-1] / b_p[ptr][t_length-1] ;
}

read_tm_file ()
{
        FILE    *tf, *fnet ;
        int     j, i, r, c ;
        int     temp, k, number ;
        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        ysize = (xsize = 10) ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < 100 ; i++) {
                                fscanf (fnet," %d %d", &map2[r][c][i][0],
                                        &map2[r][c][i][1]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                if (map2[r][c][i][0] > -1)
                                        number++ ;
                                }
                        mask[r][c] = number ;
                        }
                }
        word_counter = 0 ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

read_word (pointer)
        int     pointer ;
{
        int     flag, r, c, i, j, k ;
        double  in[16], d1, d2, d3, d4, d5 ;
        float   element ;
        int     loc2[2000][2], loc3[2000][2] ;
        FILE    *fsnd ;
        int     sound, point, x[5], y[5] ;
        int     max_pts ;
        double  max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = -1) ;
}

f_mindist (f_map, inp, close)
        double  inp[16] ;
        int     close[2] ;
        float   f_map[20][20][16] ;
{
        int     r, c, i ;
        double  dist ;
        double  minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < f_number_inputs ; i++)
                                dist += pow (inp[i] - f_map[r][c][i], 2.0) ;
                        if (dist < minimum) {
                                minimum = dist ;
```

```
                                        close[0] = c ;
                                        close[1] = r ;
                                        }
                              }
                    }
}
```

```
$ link coderb,options_file/opt
/*
******************************** coderb.c ********************************

        This routine creates an untrained second Kohonen net from a *.dat
        file of stored trajectories and tests it with a specified set of
        digits using TAXI distance.

        Trajectories are 100 x-y pairs filled with trailing -1's. The
        user may input the length used in TAXI distance.
        *************************************************************
                                        G. BARMORE25 Sep 88
*/

# include math
# include stdio
# include stat

        int             number_discretes ;
        int             word_counter ;
        int             num_words ;
        char            word_number[100][15] ;

        int             f_ysize, f_xsize, f_number_inputs ;
        float           f_map[20][20][16] ;

        float           map[20][20][225] ; /* output nodes */
        double          input[225] ; /* input nodes */
        double          node_dist ;

        int             closest[2] ; /* closest node */
        int             xsize, ysize ; /* Size of array */
        int             number_inputs ;

        char            training_file[30], temp_file[30], first_net_file[30] ;
        char            net_file[30] ;

        int             mask[20][20] ;
        int             map2[20][20][100][2] ;
        float           aa = 0.75 ;
        float           bb = 0.75 ;
        int             length[200] ;
        int             location[2000][2] ;

mindist (close)
        int             *close ;
{
        int             i, r, c ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        dist = 0.0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                dist+=abs(map2[r][c][i][0]-location[i][0])+
                                abs(map2[r][c][i][1]-location[i][1]) ;
                                }
                        dist /= length[0] ;
                        if (dist < minimum) {
```

```
                                        minimum = dist ;
                                        *close = c ;
                                        }
                                }
                        }
        node_dist = minimum ;
}

main ()
{
        int         c ;

        printf ("\nCODERb: Create codebook using x-y pairs/taxi...\n") ;
        map_speech () ;
}

map_speech ()
{
        int         r, c, i, j, k ;
        char        name_trj[30], temp[30] ;
        int         loc ;
        FILE        *fnet, *flog, *fmask ;

        printf ("\nEnter name of pre-processor Koh net-file [less .net]: ") ;
        scanf ("%s", temp) ;
        sprintf (first_net_file, "%s.net", temp) ;

        printf ("Enter name of header file containing words (less .hdr): ") ;
        scanf ("%s", temp_file) ;
        sprintf (training_file, "%s.hdr", temp_file) ;

        printf ("Enter .dat file for net generation (less .dat): ") ;
        scanf ("%s", temp_file) ;
        sprintf (net_file, "%s.dat", temp_file) ;

        printf ("Enter [number_inputs <= 100]: ") ;
        scanf ("%d", &number_inputs) ;

        read_trn_file () ;

        flog = fopen ("temp.log","w") ;
        fprintf (flog, "CODER:\n") ;
        fprintf (flog, "%s -> %s -> %s\n", training_file,
                        first_net_file, net_file ) ;
        fprintf (flog, "Size is %d by %d nodes\n", xsize, ysize) ;

        printf ("\nExpect %d calculations.\n", num_words) ;
        fprintf (flog, "Expect %d calculations.\n", num_words) ;
        for (r = 0 ; r < num_words ; r++) {
                getin () ;
                mindist (&loc) ;
                printf ("Word %3d is: %3d\n", r, loc) ;
                fprintf (flog, "Word %3d is: %3d\n", r, loc) ;
                }
        printf ("\nCalculations finished.\n") ;
        fclose (flog) ;
}

read_trn_file ()
{
        FILE            *tf, *fnet ;
        int             j, i, r, c ;
        int             temp, k, number ;


        tf = fopen (training_file, "r") ;
        fscanf (tf, "%d", &num_words) ;
        for (i = 0 ; i < num_words ; i++)
                fscanf (tf, "%s", word_number[i]) ;
        fclose (tf) ;

        fnet = fopen (first_net_file, "r") ;
        fscanf (fnet,"%d %d %d", &f_ysize, &f_xsize, &f_number_inputs) ;
        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize : c++) {
                        for (i = 0 ; i < f_number_inputs ; i++) {
                                fscanf (fnet," %f", &f_map[r][c][i]) ;
                                }
                        }
                }
        fclose (fnet) ;

        ysize = (xsize = 10) ;
        number_inputs = 100 ;

        fnet = fopen (net_file, "r") ;
        fscanf (fnet, "%d", &num_words) ;
        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        for (i = 0 ; i < 100 ; i++) {
                                fscanf (fnet," %d %d", &map2[r][c][i][0],
                                        &map2[r][c][i][1]) ;
                                }
                        }
                }
        fclose (fnet) ;

        for (r = 0 ; r < ysize ; r++) {
                for (c = 0 ; c < xsize ; c++) {
                        number = 0 ;
                        for (i = 0 ; i < number_inputs ; i++) {
                                if (map2[r][c][i][0] > -1)
                                        number++ ;
                                }
                        mask[r][c] = number ;
                        }
                }
        word_counter = 0 ;
}

getin ()
{
        if (word_counter == num_words)
                word_counter = 0 ;
        read_word (word_counter) ;
        word_counter++ ;
}

read_word (pointer)
        int             pointer ;
```

```
{
        int             flag, r, c, i, j, k ;
        double          in[16], d1, d2, d3, d4, d5 ;
        float           element ;

        int             loc2[2000][2], loc3[2000][2] ;
        FILE            *fsnd ;
        int             sound, point, x[5], y[5] ;
        int             max_pts ;
        double          max ;

        fsnd = fopen (word_number[pointer], "r") ;
        sound = 0 ;
        i = 0 ;
        flag = 0 ;
        while (flag != 1) {
                fscanf (fsnd, "%f", &element) ;
                if (feof(fsnd) !=0)
                        flag = 1 ;
                else if (i > 99)
                        flag = 1 ;
                else {
                        in[i] = (double) element ;
                        i++ ;
                        }
                if ((i == 15) && (flag == 0)) {
                        f_mindist (f_map, in, &loc2[sound][0]) ;
                        i = 0 ;
                        sound++ ;
                        }
                }
        fclose (fsnd) ;
        for (i = 0 ; i < sound ; i++) {
                loc3[i][0] = loc2[i][0] ;
                loc3[i][1] = loc2[i][1] ;
                }
        max_pts = sound ;

        ... Trajectory Reduction ...

        for (i = 0 ; i < j ; i++) {
                location[i][0] = loc2[i][0] ;
                location[i][1] = loc2[i][1] ;
                }
        length[0] = j ;
        for (i = j ; i < 200 ; i++)
                location[i][1] = (location[i][0] = -1) ;
}

f_mindist (f_map, inp, close)
        double          inp[16] ;
        int             close[2] ;
        float           f_map[20][20][16] ;
{
        int             r, c, i ;
        double          dist ;
        double          minimum = 9.99e31 ;

        for (r = 0 ; r < f_ysize ; r++) {
                for (c = 0 ; c < f_xsize ; c++) {
```

```
            dist = 0.0 ;
            for (i = 0 ; i < f_number_inputs ; i++)
                dist += pow (inp., - f_map[r][c][i], 2.0) ;
            if (dist < minimum) {
                minimum = dist ;
                close[0] = c ;
                close[1] = r ;
                }
            }
        }
}
```

Appendix C. Software User's Manual

Introduction

The purpose of this user's manual is twofold. First, it will demonstrate how to use the "C" programs developed by Capt Barmore for his thesis, *Speech Recognition Using Neural Nets and Dynamic Time Warping*. Secondly, it provides adequate information for someone with a raw copy of the backup tape containing source, executable, and sound files to modify and recompile that information on his own system.

Since this manual is written as an appendix to the subject thesis, it does not have a table of contents or index. Thus, it is suggested that the reader skim the manual before beginning to read in depth. The section headings should provide adequate information about the manual's layout.

Please note that no attempt is made to describe all of the programs generated for this thesis or even all of the options within the programs described below. When an option or program is not described, its function can usually be determined by the input parameter prompt or the comments in the source code, respectively. Also, the author does not guarantee the performance or use of any program provided and is not liable for any damage pursuant to the use of any of these programs. Finally this code was developed under Air Force funding and is therefore the property of the United States Air Force.

The Backup Tape

The backup tape is simply that—a copy of all of the files used in the subject thesis that were stored on the VAX 11/780 (with node-name I780A). The author is not an expert on backups and magnetic tape format; thus, the only information available is that the data was transferred using the "backup" utility under the VMS operating system. The directories were "purged", but no attempt was made to remove data or program files that were either superceded or just not useful.

The sections immediately following this paragraph provide a general description of the contents of each of the directories. Sample runs of the most significant programs are provided in later sections. The names of the following subsections actually correspond to the names of the directories—or rather, that part of the path name corresponding to the subdirectories beneath [KABRISKY.GBARMORE].

Dev. This directory contains the source (*.c), object (*.o), executable (*.exe), and *.com files generated during program development. Not all of the *.o and *.exe files were saved. However, they can be regenerated by compiling the appropriate source file (*.c) with a "cc filename" command, or by relinking the object files using the respective *.com files in a "@filename" command. Notice that the *.com files contain the link command for the appropriate program. Also, if the user is not on a VMS system, he may have to modify the two commands just described.

When recompiling on a VMS system, the following two commands should be executed prior to the "cc" command:

```
DEFINE LNK&LIBRARY SYS$LIBRARY:VAXCCURSE.OLB
DEFINE LNK&LIBRARY_1 SYS$LIBRARY:VAXCRTL.OLB
```

The "include" commands in each source file should also be examined for correct format. VMS does not (in most cases) use the UNIX "<*.h>" format. Additionally, some programs use GKS graphics. The source files for these programs contain the line "# include <gksdefs.h>". For these programs to run effectively, the workstation used should have at least the capability of a Micro-VAX II.

Lin. Most of the work performed in this thesis used sound files sampled and digitized using a logarithmic sampling scheme to increase dynamic range. However, tests were run to see if there was any difference when normal, *linear* sampling was used. This directory contains the files used for those tests.

Net. A number of small sound files that approximate "pure" phonemes are found in this directory. These were used to see where a particular sound was placed in the first Kohonen net.

Snd2. This directory contains an assortment of *.trn files (generated by the AUTOFFT routine with a *.snd file as input) used in the early development and testing of the DTW routines. Notice that the Kohonen nets in this directory are named in the "speech*.net" series. This series was an early version of the first Kohonen net and is not used in any of the results reported in this thesis.

Snd3. The files in this directory are used to perform AUTODTW runs. Notice that the first Kohonen net files in this directory are the named in the final series, "speak*.net".

Sndw. Most of the sound files used in this thesis were sampled with a resolution of eight bits per sample on an ATARI microcomputer. The files in this directory were sampled using a 16 bit A/D converter. Only limited tests were performed on these files.

Sounds. The files in this directory are the sound (*.snd) files from which the *.trn files were generated that trained both the first and second level Kohonen nets as well as testing the first Kohonen nets. These files include a large number of isolated words identified by "*word*.snd" where "word" is replaced by one choice from the following set: {one, two, three, ..., nine, silence}. The content of the files should be obvious.

Also included in this directory are several continuous speech utterances. These include the first net training utterances, usually identified as "*dig*.snd", and connected utterances used in AUTODTW tests (either "b*.snd" or "g*.snd"). The "*dig*.snd" files contain the sequence of digits zero through nine spoken in order, usually with short pauses between the digits. The background noise (silence) is a quiet computer hum.

The "b*.snd" and "g*.snd" files contain selected sequences of digits that were chosen to correspond to the utterances used by Capt Dawson in his thesis (13) the preceding year. The contents of each file can be determined simply by examining how the "*" is replaced in the file name. For example, if the file is "b282828.snd", it contains the sequence of words 2-8-2-8-2-8. The "b*.snd" series were spoken clearly and distinctly. The "g*.snd" series were spoken in a fast, slurred fashion.

Sounds2. This directory contains the sound files from which the *.trn files were generated to test the second Kohonen nets. It also contains the "m*.snd" series of connected utterances. These *connected* utterances were spoken by a different speaker than all of the other utterances. The "m*.snd" series were spoken by a female with a relatively deep voice, while all of the other utterances are in a mid-range male voice. See the preceding section entitle "Sounds" for the naming conventions.

Src. This directory was an early attempt to separate the source files (*.c) for shorter, limited backups. The plan was abandoned early, and all of these files are duplicated in the directory [.DEV].

Test. These files were used in the first tests of the AUTODTW program.

Programs

Each of the programs described below are found in the directory [.DEV]. The preceding subsection, "Dev", should be referenced for comments on how to recompile the programs. It is recommended that most of these programs be run in the batch mode. With few exceptions, any of these programs will run from 1.5 to 12 hours given typical input parameters and a Micro-VAX workstation.

Each program description states the purpose of the program, the required inputs, and the form of the output. A sample interactive session is included where it is appropriate, and when necessary, additional comments are made.

Autofft. The program AUTOFFT converts a sound file (*.snd) into a file (*.trn) containing a sequence of 15 component vectors. The *.snd file is binary, has eight bytes of header information, and contains an unspecified number of sound samples (one sample per byte). In the sound file, the binary integers from −128 to −1 are mapped into the range +128 to +255. Basically, AUTOFFT performs the preprocessing functions on the sound waveform.

To simplify the user's effort in preprocessing a large number of files, AUTOFFT reads the file "sounds.hdr" to obtain the names of the *.snd files to preprocess. An example of "sounds.hdr" is:

```
I780A>type sounds.hdr
2
lzero
six2
I780A>
```

Notice that the "2" corresponds to the number of files to be processed, and that each filename does *not* include the three letter file-type. Files "lzero.snd" and "six2.snd" must be present in the current directory or an error will result when AUTOFFT is executed. The following is a sample run:

```
I780A>run autofft
FFT3:  Time/Frequency Conversion for Kohonen Net ...
Enter (0) logarithmic, or (1) Kohonen reduction:  0
     lzero.trn opened ... 84 vectors.
     six2.trn opened ... 63 vectors.
I780A>
```

The type of reduction (from 128 to 15 components) described in the thesis is (0) logarithmic. The (1) Kohonen style reduction approximates the methods described in Kohonen's articles, but does not work very well since no corresponding filter is used.

Neural7. Once a training file (*.trn) is preprocessed using AUTOFFT, it can be used to train the first level Kohonen neural net with the program NEURAL7. The net created with NEURAL7 is a two dimensional net limited to no more than 20 nodes in either direction (400 nodes total). The program is "hardwired" for 15 inputs (a 15 component input vector) and a conscience factor of

$\beta = 1.5$. The conscience factor is easily changed since it is a global variable set at the beginning of the "neural7.c" file. Simply change the value and recompile. Alternatively, one can easily modify the program to make β an input variable (by changing the subroutine "userinp" in "neural7.c").

A sample run shows the input parameters and the status information:

```
I780A>run neural7
NEURAL7 (Net training with conscience only!) ...
Enter size 'm n' (for an m x n) of array = ? [int int]  15 15
Enter name of training file [.trn assumed]:  ldig3
     Training file is:  ldig3.trn
Enter name of net file to create [.net appended]:  speak1
     Net file to be created:  speak1.net
Number of iterations = ? [int]  90000
Number of iterations between status messages = ? [int]  45000
For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR :  2
First segment starting gain = ? [float]  .1
Second segment starting gain = ? [float]  .01
Second segment starting iteration = ? [float]  20000
Do you want 0) sequential or 1) randomized training?  0
Starting size of neighborhoods 'yn xn' = ? [int int]  7 7
Final size of neighborhoods 'yn xn' = ? [int int] 1 1
Initial seed for random # generator (0 SELECTS TIME) = ? [int]  33
Ready to begin? (y/n)  y

NEURAL3: gain=0.006428, yrange=1, xrange=1, iteration #45000 (of 90000)
37 ave nodes eliminated
NEURAL3: gain=0.000000, yrange=1, xrange=1, iteration #90000 (of 90000)
36 ave nodes eliminated
Net file:  speak1.net saved!
I780A>
```

Most of the questions are self explanatory, but the request for a gain curve is not. The gain curve can be either linear, sigmoidal, or piecewise linear (hardwired for two segments). The linear gain curve is simply a linear (continuously decreasing) change in gain. A sigmoidal curve starts high and ends low, but such a gain curve (as implemented here) does not seem to properly train a net. The piecewise linear curve works well, with both gain segments ending in zero gain.

The application of 15 component training vectors to the inputs of the net can be either sequential or random. In the former, the first vector in the file is applied first, and the rest in order. If the number of training iterations required is greater than the number of vectors in the file, the sequence of vectors is simply repeated. Random training picks a vector from the training file at random for each application of an input.

The size of the training neighborhood is changed in a linear fashion. The neighborhood is specified by a value, r (actually two such values), such that the size of the neighborhood in the respective direction is $2r + 1$ nodes. Thus a starting neighborhood of "7 7" is actually a "15 x 15" array of nodes.

The seed for the random number generator should be selected by the user. The code for obtaining the seed by examining the time still has a bug in it; also, specifying a seed allows reproduction of the resulting net.

Finally, note the program title is not correct in the status message (an oversight), and the "results" of conscience are reported. The average number of nodes eliminated is the average number of nodes, per training iteration in the reporting period, not eligible to "light up" (as the center node in the neighborhood).

Neural2. This program displays a first Kohonen net graphically. Each node is represented by 15 vertical bars in a small spectrum-like graph. The 15 bars correspond to the 15 positive and negative weights. The spectra are shown in an array corresponding to the size of the actual net. GKS graphics routines are used. The following is a sample run:

```
I780A>run neural2
NEURAL2 (Sound net Training with GRAPHICS only!) ...
Do you want to train a net? (y/n)  n
Do you want to draw spectra of a net? (y/n)  y
Enter name of net-file to test:  speak1.net
Was this generated by (0) FFT2 or (1) FFT3 ?  1
... graphics displayed now ...
I780A>
```

Notice that you can also train a net (without conscience); however, it is much slower since graphics are provided as status messages. The current version of preprocessing, incorporated into AUTOFFT, corresponds to FFT3. After the graphics display is complete, the program is exited by the user entering any string followed by a carriage return. *This is the standard way to leave all graphics displays in these programs!*

Neural4. This program permits the user to graph trajectories through the first Kohonen net. Again GKS graphics routines are used. The following is a sample run:

```
I780A>run neural4
NEURAL4 (Sound TRAJECTORIES only!) ...
Do you want (0) sound file created or (1) not?  1
Do you want (0) NO graphics or (1) TRAJECTORIES:  1
Enter name of net-file to use [less .net]:  speak1
Enter name [next] of speech file to map [less .trn]:  zero3
... full trajectory graphed here ...
1
... reduced trajectory graphed here ...
1
Enter name [next] of speech file to map [less .trn]:  zero6
... full trajectory graphed here ...
1
... reduced trajectory graphed here ...
^C
I780A>
```

Notice that a *.snd file can be regenerated from the trajectory (if you know what file was used to train the net and if the training utterance's *.snd and *.trn files are located in the current directory). However, creating a sound file is slow and incorporates distortion from various sources. If no graphics are desired, an ASCII file (*.trj) is created (it's actually created in either case) which contains the various steps in the trajectory reduction process. Each graphics display is left by entering a string (here it's "1") and a carriage return.

Autodtw. This program uses dynamic time warping to test a set of standard utterances. Trajectories are created for a set of templates, which in turn are compared with the trajectories created for the set of standard utterances. The templates are listed in a file such as "t2.hdr":

```
I780A>type t2.hdr
11
1zero3.trn
1one3.trn
1two3.trn
```

```
lthree3.trn
four5.trn
lfive3.trn
six8.trn
lseven3.trn
leight3.trn
lnine3.trn
silence3.trn
I780A>
```

A file called "standard.hdr" which contains the names of the files listing the standard utterances must be in the current directory. For example:

```
I780A>type standard.hdr
4
iso.hdr
Speaker_Dependent_Isolated_Words
con.hdr
Speaker_Dependent_Connected_Words
indiso.hdr
Speaker_INDEPENDENT_Isolated_Words
indcon.hdr
Speaker_INDEPENDENT_Connected_Words
I780A>
```

The strings following each file name in standard.hdr are printed out during an AUTODTW run prior to the tests on that file's set of utterances. Scoring is performed automatically in AUTODTW, and cumulative scores are reported for each file's set of utterances. An example of a set of utterances is:

```
I780A>type iso.hdr
5
kzero1.trn 1
0
kone1.trn 1
1
ktwo1.trn 1
2
kthree1.trn 1
3
ldig3.trn 10
0 1 2 3 4 5 6 7 8 9
I780A>
```

Notice that the first number in the file is the number of utterances to test. Then comes the file names of each utterance followed by the number of digits in the respective utterance and what those digit(s) are.

With all of the above files in the current directory, along with a trained net, AUTODTW can be run:

```
I780A>run autodtw
AUTODTW:  Tests standard set of utterances ...
Enter name of template file [less .hdr]:  t2
Enter name of log file [add .log]:  temp.log
Enter horizontal weight:  .75
Enter vertical weight:    .75
Enter name of net to use [less .net]:  speak1
     lzero3.trn is 75 vectors long
     lone3.trn is 65 vectors long
     ltwo3.trn is 52 vectors long
     lthree3.trn is 68 vectors long
     four5.trn is 69 vectors long
     lfive3.trn is 52 vectors long
     six8.trn is 65 vectors long
     lseven3.trn is 70 vectors long
     leight3.trn is 37 vectors long
     lnine3.trn is 82 vectors long
     silence3.trn is 16 vectors long

Speaker_Dependent_Isolated_Words
     kzero1.trn   is:  0
          Should be:  0
     correct = 1.000     cum_correct = 1.000
     koen1.trn    is:  1
          Should be:  1
     correct = 1.000     cum_correct = 1.000

     . . .

     ldig3.trn    is:  0 1 2 3 .. 4 5 6 7 .. 8 9
          Should be:  0 1 2 3 4 5 6 7 8 9
     correct = 1.000     cum_correct = 1.000

Speaker_Dependent_Connected_Words

     ...  results continue ...
I780A>
```

The weights requested are the stretch factors used in the DTW routine. In addition to printing the results on the terminal's screen (about five seconds per digit), the results are written to the log file, "temp.log". The ".." seen in the output represents silence. Note that substitutions and deletions count as one error while insertions only count as half an error (this is hardwired in the scoring DTW routine's stretch factors). Also be aware that there are unusual cases where the scoring will make a mistake on isolated speech. Check all results by examining the log file.

Outdat4. To save time in training the second level Kohonen nets, a set of trajectories is precalculated and stored (for later use in TWOBAS4) by OUTDAT4. The list of utterances, from which the trajectories are made, is kept in a header file. For example:

```
I780A>type twokoh2.hdr
100
zero0.trn
one0.trn
two0.trn
three0.trn
four0.trn
. . .
five9.trn
six9.trn
seven9.trn
eight9.trn
nine9.trn
I780A>
```

In the example shown, 100 trajectories are created, 10 of each of the 10 digits. The actual run might be:

```
I780A>run outdat4
OUTDAT4:  Prepare training data [x,y], second kohonen ...
Enter name of header file containing words (less .hdr):  twokoh2
Enter name of pre-processor Kohonen net file [less .net]:  speak1
Enter name of data file to create [less .dat]: qpath
I780A>
```

The trajectories are stored in the ASCII file, "qpath.dat".

Twobas4. This program trains the second Kohonen net and requires input parameters very similar to NEURAL7. The biggest differences are the precalculated trajectories (isolated digits) and user selectable conscience factor (β). A sample run is:

```
I780A>run twobas4
TWOBAS4: Train 2nd Koh with 2-D trajectories ...
Enter conscience factor (> 1.0): [float]  1.5
Enter size 'm n' (for an m x n) of array = ?  [int int]  10 10
Do you want 0) sequential training,
            1) randomized training?  0
Enter name of training file [less .dat]:  qpath
Enter name of net file to create [less .net]:  path21
Number of iterations = ? [int]  150000
Number of iterations between status messages = ? [int]  10000
For gain enter 0) LINEAR, 1) SIGMOIDAL, 2) PIECEWISE LINEAR :  2
First segment starting gain = ? [float]  .1
Second segment starting gain = ? [float]  .01
Second segment starting iteration = ? [float]  30000
Starting size of neighborhoods 'yn xn' = ? [int int]  4 4
Final size of neighborhoods 'yn xn' = ? [int int] 1 1
Initial seed for random # generator = ? [int] 33
Ready to begin?  (y/n)  y

TWOBAS4: gain=0.066669, yrange=3, xrange=3, iteration#10000 (of 150000)
. . .
TWOBAS4: gain=0.000000, yrange=1, xrange=1, iteration#150000 (of 150000)
Net file:  path21.net saved!
I780A>
```

Twomask5. Since a lot of the "trailing" weights at the end of any node's weight vector are −1's, a *.msk file is created by TWOMASK5 (where "*" corresponds to the "*" in the net file "*.net" created by TWOBAS4). The *.msk is an ASCII file used in the TWOPIC series of programs to identify the effective length of the trajectories represented by each node's weight vector. Running the program is simple:

```
I780A>run twomask5
TWOMASK4 (Creates net mask for 2-D trajectories
) ... Enter name of output Koh net_file [less .net]:  path11
...  status information ...
I780A>
```

Obviously, correcting prompt messages had a low priority in the thesis work.

Twopic4c. This program shows graphically which nodes light up (using Euclidean distance) when a set of isolated digits are applied to a second Kohonen net. This program uses GKS graphics routines. For example:

```
I780A>run twopic4c
TWOPIC4c (Plot Words for 2-D Reduced Queued Traj) ...
Enter name of pre-processor Koh net-file [less .net]:  speak1
Enter name of header file containing words (less .hdr):  twokoh2
Enter name of output Koh net_file [less .net]:  path11
... graph net ...
1
I780A>
```

Notice that the header file containing the utterances has the same format as that used in OUTDAT4. Again, a string and carriage return exit the graphics display.

Twopic8b. This program finds the appropriate digit, with which to label each node in a second Kohonen net, by looking through a list of trajectories and finding the closest one (by a mini-DTW) to each node's weight vector. Again, the file listing the utterances is in the same format used by OUTDAT4. A sample run is:

```
I780A>run twopic8b
TWOPIC8b (Closest word for each node: 100 wts/2-D )...
Enter name of pre-processor Koh net-file [less .net]:  speak10
Enter name of header file containing words (less .hdr):  test3
Enter name of output Koh net_file [less .net]:  path22
Expect 100 calculations

Reading word: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
95 96 97 98 99

Node  0 : word # 22, dist = 9.465583e-01  (dtwo2.trn)
. . .
Node 99 : word # 87, dist = 0.423736e-00  (deight7.trn)
Calculations finished.
I780A>
```

The output is written to "temp.log" in addition to being displayed on the terminal's screen. Notice how the program "labels" a node—by giving the user the closest trajectory. In this example, node 99 is identified as an eight. The output of this program should be saved for use in TWOPIC6b.

Twopic6b This program takes the node labels determined by TWOPIC8b and tests a set of unknown utterances (using mini-DTW as the distance algorithm). Unfortunately, each time TWOPIC8b is run and the labels are changed, the file "lookup6.c" must be modified and recompiled and then TWOPIC6b relinked. This process could be modified with code changes to both TWOPIC8b and TWOPIC6b. But for now, this is the portion of "lookup6.c" that must be changed to reflect the results of TWOPIC8b:

```
I780A>type lookup6.c
. . .
     int  look_up[100] = {9,9,9,3,7,7,7,6,6,6,
                          9,9,3,3,3,3,6,6,6,6.
                          9,9,3,2,3,0,6,6,6,6,
                          1,1,1,3,3,0,3,6,6,6,
                          1,1,1,3,0,0,0,4,4,4,
                          3,1,9,3,0,0,0,0,4,4,
                          5,3,3,3,3,3,7,0,4,0,
                          5,5,2,3,2,7,7,7,8,2,
                          5,5,2,2,3,3,7,4,8,8,
                          5,5,9,3,0,3,4,3,8,8} ;
. . .
I780A>
```

In this lookup table, each row of integers corresponds to the digits labelling a row of nodes. The first node is the first integer, the last node is the 100^{th} integer. Note that the program is hardwired for a 10 by 10 Kohonen neural net. The update process is:

```
I780A>edit lookup6.c
. . .
I780A>cc lookup6
I780A>@twopic6b
I780A>
```

In this case twopic6b.com consists of:

```
I780A>type twopic6b.com
```

```
$ del twopic6b.exe
$ link twopic6b,nwin5b,lookup6,options_file/opt
I780A>
```

Now, the user is ready to run TWOPIC6b to actually test the speech recognition capability of a second Kohonen net. For example:

```
I780A>run twopic6b
TWOPIC6B (DTW Words for 2-D Reduced Queued Traj) ...
Enter name of pre-processor Koh net-file [less .net]:  speak10
Enter name of header file containing words (less .hdr):  test3
Enter name of output Koh net_file [less .net]:  path21
Expect 100 calculations.

0 :  [0,4]  dist = 1.350917e+00    zero
. . .
99 :  [8,6]  dist = 0.938755e-01    nine
Calculations finished.
I780A>
```

Notice that each word from "test3.hdr" is identified by a sequential number (here 0 through 99) according to its order in "test3.hdr". The node that lit up is in braces, and the digit the utterance was found to be is spelled out in the far right column. Scoring this output requires knowing what word is in what position in "test3.hdr". To simplify the scoring procedure, the author (almost) always used 10 examples of each of the 10 digits *in order.*

Twopic8c. This is a later version of TWOPIC8b where the mini-DTW distance algorithm is changed to a TAXI distance.

Twopic6c. This is a later version of TWOPIC6b where the mini-DTW distance algorithm is changed to a TAXI distance. In this case, "lookup7.c" (instead of "lookup6.c") should be updated with the results from TWOPIC8c.

Coder. This program creates (and tests) an "untrained" second Kohonen net from a file of precalculated trajectories (created by OUTDAT4). The trajectories are transferred directly to the nodes' weight vectors without any training. Tests are run on the untrained net for a set of test utterances. The distance algorithm is a mini-DTW. The following is a sample run:

```
I780A>run coder
CODER:  Create codebook using x-y pairs/dtw ...
Enter name of pre-processor Koh net-file [less .net]:  speak1
Enter name of header file containing words (less .hdr):  test1
Enter .dat file for net generation (less .dat):  qpath
Enter [number_inputs] desired (<=100):  75
Expect 100 calculations.

Word  0 is:  0
Word  1 is:  0
 . . .
Word 99 is:  9
Calculations finished.
I780A>
```

The far right column corresponds to the calculated content of the word being tested. It assumes that "qpath.dat" was generated from 100 words, 10 of each digit in sequence. For simplicity in scoring, the author placed a similar sequence of digits (using other examples) in "test1.hdr". Notice that the number of inputs is variable in this program.

Coderb. This program is the same as CODER, except that the mini-DTW distance algorithm is changed to a TAXI algorithm.

Summary

The above paragraphs describe both the contents of the backup tape and the most significant programs found on that tape. Further questions can be answered by contacting the author or perusing the source code found in the directory [.DEV] or Appendix B of this thesis.

Bibliography

1. Wallich, Paul. "Putting Speech Recognizers to Work," *IEEE Spectrum, 24:* 55–57 (April 1987).
2. Doddington, George R. and Thomas B. Schalk. "Speech Recognition: Turning Theory to Practice," *IEEE Spectrum, 18:* 26–32 (September 1981).
3. Levinson, Stephen E. and Mark Y. Liberman. "Speech Recognition by Computer," *Scientific American, 244:* 64–76 (April 1981).
4. Routh, Capt Richard L., USA. *A Spoken English Recognition Expert System*. MS thesis, AFIT/GCS/EE/83S-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1983.
5. Rothfeder, Jeffrey. "A Few Words About Voice Technology," *PC Magazine, 5:* 191–205 (30 September 1986).
6. Kohonen, Teuvo. "The 'Neural' Phonetic Typewriter", *Computer, 21:* 11–22 (March 1988).
7. Lippmann, Richard P. "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine, 4:* 4–22 (April 1987).
8. Kohonen, Teuvo and others. "Phonotopic Maps—Insightful Representation of Phonological Features for Speech Recognition," *Proceedings of the Seventh International Conference on Pattern Recognition*. 182–185. Los Angeles: IEEE Computer Society, 1984.
9. Kohonen, Teuvo. "Dynamically Expanding Context, with Applications to the Correction of Symbol Strings in the Recognition of Continuous Speech," *1986 International Conference on Pattern Recognition*. 1148-1151. Los Angeles: IEEE Computer Society Press, 1986.
10. Ludeman, Lonnie C. *Fundamentals of Digital Signal Processing*. New York: Harper & Row, Publishers, 1986.
11. O'Neill, Mark A. "Faster Than Fast Fourier," *Byte, 13:* 293–300 (April 1988).
12. Ney, Hermann. "The Use of a One–Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 32:* 263-271 (April 1984).
13. Dawson, Capt Robert G. *Spire Based Speaker-Independent Continuous Speech Recognition Using Mixed Feature Sets*. MS thesis, AFIT/GE/ENG/87D-14. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
14. Rabiner, Lawrence R. "Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 26:* 575-582 (December 1978).
15. Juang, Biing-Hwang and Lawrence R. Rabiner. "Mixture Autoregressive Hidden Markov Models for Speech Signals," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 33:* 1404–1413 (December 1985).
16. Hussain, Capt Ajmal, PAF. *Limited Continuous Speech Recognition by Phoneme Analysis*. MS thesis, AFIT/GE/EE/83D-31. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983. (A138 021)
17. Dixon, 1Lt Kathy R. *Implementation of a Real-Time, Interactive, Continuous Speech Recognition System*. MS thesis, AFIT/GE/ENG/84D-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.
18. Montgomery, 2Lt Gerard J. *Isolated Word Recognition Using Fuzzy Set Theory*. MS thesis, AFIT/GE/EE/82D-74. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982. (A124 851)

19. Kohonen, Teuvo. "Learning Vector Quantization and the K-Means Algorithm," *1988 International Conference on Neural Networks, Tutorial # 10, Self-Organizing Feature Maps, Appendix 4*: 1–2. San Diego: IEEE Computer Society Press, 1988.

20. Kim, Capt Peter Y. *F-16 Speaker-Independent Speech Recognition System Using Cockpit Commands (70 Words)*. MS thesis, AFIT/GE/ENG/88D-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

Vita

Captain Gary D. Barmore [redacted] received a Bachelor of Science degree in Physics from Purdue University in May 1978. He spent two years as an engineer for McDonnell-Douglas Astronautics before entering Washington University School of Law. In 1981 he left law school to enter the USAF. He received his commission through OTS in February 1982, and then attended Louisiana Tech University in an AFIT/CI program. He received a Bachelor of Science in Electrical Engineering from Louisiana Tech in May 1983. His follow-on assignment was at Space Division (AFSC) in the Navstar/Global Positioning System Joint Program Office where he became chief of the Satellite Systems Division. He entered the School of Engineering, Air Force Institute of Technology, in June 1987.

[redacted]

ADA202528

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GEO/ENG/88D-1

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State, and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

7b. ADDRESS *(City, State, and ZIP Code)*

12 Jan 1989

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Speech Recognition Using Neural Nets and Dynamic Time Warping UNCLASSIFIED

12. PERSONAL AUTHOR(S)
Gary D. Barmore, Capt, USAF

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM ________ TO ______

14. DATE OF REPORT *(Year, Month, Day)*
1988 December

15. PAGE COUNT
297

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
25	04	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Speech, Speech Recognition, Neural Nets, Dynamic Time Warping, Kohonen Neural Nets

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

Thesis Chairman: Dr. Matthew Kabrisky, PhD, Professor of Electrical Engineering

A speech recognition system is described that uses neural nets and dynamic time warping to recognize speaker independent, isolated and connected speech. The system uses a Kohonen neural net to characterize an utterance as a trajectory through a two dimensional space. The trajectories are input to a word recognition algorithm---either one pass dynamic time warping (DTW) or a modified second Kohonen neural net---to determine the content of the utterance. For a small vocabulary consisting of the digits zero through nine, DTW correctly identifies up to 99% of isolated speech and 93% of connected speech. On the same vocabulary, the modified second Kohonen neural net correctly identifies up to 96% of isolated speech. The second Kohonen net processing was not designed to efficiently identify connected speech.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
□ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Matthew Kabrisky, GS-15

22b. TELEPHONE *(Include Area Code)*
(513) 255-5276

22c. OFFICE SYMBOL
AFIT/ENG