1991

Vowel recognition using Kohonen's self-organizing feature maps

Anand R. K. Sundaram

Rochester Institute of Technology
Computer Science Department

Vowel Recognition Using Kohonen's Self-organizing Feature Maps

by

Anand R.K. Sundaram

Thesis, submitted to

The Faculty of the Computer Science Department

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Approved by:

Dr. Peter G. Anderson

Professor Robert T. Gayvert

Professor John A. Biles

July 25, 1991

I, Anand R.K. Sundaram, prefer to be contacted each time a request for a reproduction is made. I can be reached at the following address:

c/o Bull HN Information Systems Inc.
300 Concord Road, MA30/826A
Billerica, MA 01821

July 25, 1991

Vowel Recognition using Kohonen's Self-organizing Feature Maps

Abstract

An important organizing principle observed in the sensory pathways in the brain is the orderly placement of neurons. Although the neurons are structurally identical, the specialized role played by each unit is determined by its internal parameters that are made to change during early learning processes. In the human auditory system, the nerve cells and fibres are arranged in a manner that would elicit maximum response from the neurons when they are activated. Although most of this organization is genetically determined, some of the high level organization is created due to algorithms that promote self-organization. Kohonen's self-organizing feature map is a neural net model that produces feature maps similar to the ones produced in the brain. These maps are capable of describing topological relationships of input signals using a one or two dimensional representation. This technique uses unlabeled data and requires no training as in supervised learning algorithms. It is hence immensely useful in speech and vision applications.

This neutral net has been implemented for the recognition of vowels in the American English language. The net has been trained and tested with vowel data. The formation of internal clusters or categories has been observed and closely reflects the tonotopic relationships between the vowels. An analysis of the results has been carried out and the performance has been compared to other classification techniques. A graphical user interface has also been developed using **Xview** to help visualize the formation of the maps during the training and testing processes.

Computing Review Codes:

I.2.7: Speech Recognition and Understanding
I.6.3: Applications (Kohonen's Neural Net)

Keywords:

Artificial Neural Networks, Speech Recognition,
Vowel Classification, Self-organizing Feature Maps,
Unsupervised Learning, Kohonen, Peterson and Barney.

Acknowledgements

I would first like to thank Dr. Peter Anderson for his guidance, comments and suggestions that helped resolve all the problems I faced over the duration of this thesis. I greatly appreciate the readiness with which he has been accommodating me into his extremely busy calendar at short notice.

I am immensely thankful to Professor Rob Gayvert of the RIT Research Corporation. He has been unstinting in giving me time and encouragement as well as steering me towards the completion of the task at hand. His interest in seeing me succeed has had a significant effect on the outcome of my efforts.

I would like to thank Professor Al Biles for having introduced me to Speech Processing and Knowledge Based Systems in a way that made them seem so interesting and easy. I would also like to thank him for his comments and suggestions that helped improve my graphical user interface.

I would like to thank Dr. Van den Bout of North Carolina State University for his suggestions regarding the implementation of the Kohonen neural net.

My thanks are also due to Mr. Paul Allen and Mr. Daryl Johnson for their having given me exclusive access to a SPARC server to run my simulations.

In a similar vein, I would like to thank Mr. Mark Tremblay (from ISC) for having given me access to Vader (the fastest server available at this time) and for having provided me other resources critical to the completion of my tests.

I would like to thank Mr. Jack Unrue, for having compiled all the Xview source at short notice and introducing me to Imake.

Last, but not the least, I would like to thank Beth Cooper, who has alternately played the role of a friend, employer, advisor and office mate, for using her expertise in helping me select colors for the graphical user interface.

Anand R K Sundaram
July 25, 1991

Table of Contents

1.0 Kohonen's Self-organizing Feature Map

1.1 Introduction to Neural Nets

Neural nets are massively parallel computational models that seek to match human performance in areas such as speech and vision. Artificial neural net models or neural nets are massively parallel computational models that have dense interconnections of simple computational elements. Neural nets are extremely useful in problem domains that require pursuit of multiple hypotheses in parallel and a high rate of computation. The parallelism is achieved by having many computational elements connected by links with variable weights. The computational elements or nodes in biological systems are nonlinear, analog and slow. Artificial neural net models are specified by the net topology, node characteristics and training or learning rules. These rules specify an initial set of weights and specify how they should be modified during use to improve performance [*Lippmann87*]. There are a number of such models that are being used as classifiers. **Kohonen's self-organizing feature map** is a neural net model that uses unlabeled data to produce feature maps. It requires no teaching as in supervised learning algorithms such as the Back Propagation model. Neural nets that can self-organize often can outperform conventional classifying algorithms. Kohonen's net, when provided with unlabeled input, forms internal clusters or categories. These clusters effectively compress data representing and summarizing the structure of the data space. These features are especially useful in speech recognition where many hypotheses need to be pursued and a large amount of data needs to be processed. The Kohonen map may be used to produce phoneme maps from speech utterances. These can be segmented to produce standard phonemic transcriptions.

1.2 Self-organization and Feature Maps in the Brain

An important organizing principle that is observed in the sensory pathways of the brain is the orderly placement of the processing units (the neurons) that reflects some physical characteristic of the stimulus being sensed. Although the processing units may be structurally identical, the specialized role played by these units is determined by their internal parameters that are made to change during early learning processes. These processes appear to produce a meaningful ordering of the neurons. In the human auditory system, neural cells and fibres are arranged in

relation to the frequency that would elicit best response from the neurons when they are activated. Although most of the low-level organization of the neurons is genetically pre-determined, it is considered likely that some of the higher level organization is created due to algorithms that promote self-organization [*Lippmann87*]. The maps that are formed from self-organization are capable of describing the topological relationships of input signals using a one or two dimensional representation.

The ability to represent data in a economic fashion without loss of any information is a major challenge in the field of information sciences, and this ability is an obvious characteristic of the way the brain operates. During the processing of information in the human brain, there is a tendency to form reduced representations of the most important facts without losing any knowledge about the interrelationships [*Kohonen88a*]. Intelligent information systems attempt to match this by creating simple images of the world at various levels of abstraction. There is evidence that feature maps exist in the human and animal brain. The auditory cortex has a tonotopic map in which the spatial order of the cell responses corresponds to the pitch or the acoustic frequency of the tones perceived. The presence of maps in the brain formed due to sensory experiences has also been measured. In experiments involving a rat [*Olton77*] that was learning to identify food and its current location in a maze, it has been seen that, after training, certain cells in the hippocampal cortex responded only when the rat was in a particular corner. There is also reported evidence of other kinds of maps existing in the hippocampus and other parts of the brain. Although the main structures of the neural network in the brain are determined genetically, there is evidence that the sensory projections in the brain are affected by experience during learning processes. For example, after ablation of sensory organs or brain tissue at an early age or after sensory deprivation, the corresponding territory of the brain has been found to be occupied by the remaining projections. Neural cells are recruited to different tasks depending on experiences which suggests the presence of self-organization that is controlled by sensory information [*Hunt75*]

It has been observed in investigations [*Kohonen88a*] that certain networks that have dense interconnections of adaptive units have the ability to change their responses in such a way that the location of the cell where the response is maximum becomes specific to a certain feature in the input vector. It must pointed out that the cells

themselves do not move anywhere. Instead, it is their internal parameters, which give them this specificity, that are made to change. These networks are predominantly two dimensional, and their mappings are able to preserve higher dimensional topological relationships while reducing the dimensionality of the representation space.

1.3 Lateral Interaction of Inter-connected Units

Most neural networks in the brain are two dimensional layers of processing units [*Kohonen88*]. There is dense interconnection of the units through lateral feedback (Figure 1). The figure represents an array of neurons, each of which receives the primary input and the lateral connections of other units in the array.

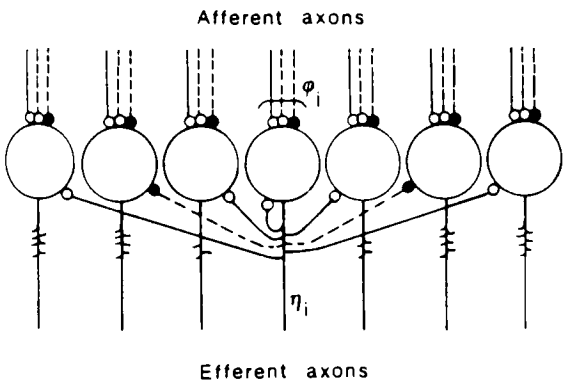


Figure 1: Lateral Interactions between neurons, (*adapted from [Kohonen88]*)

There is both anatomical and physiological evidence indicating the existence of the following types of lateral interactions between cells in mammals:

(1) Short-range lateral excitation reaching up to a radius of 50 to 100μm.

(2) A penumbra of inhibitory action reaching up to a radius of 200 to 500 μm

(3) A weaker excitatory action surrounding the inhibitory penumbra reaching up to a radius of several centimeters.

The degree of lateral interaction is considered to be of the type of a Mexican Hat (Figure 2).

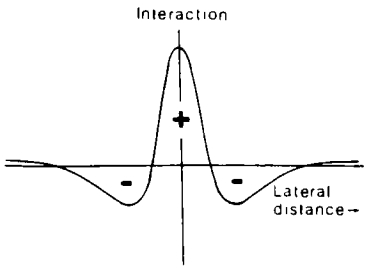


Figure 2: The Mexican Hat function of lateral interaction, *(adapted from [Kohonen88])*

1.4 Topology Preserving Mapping

In the formation of self-organizing feature maps, the aim is to transform an input signal of arbitrary dimensionality onto a one or two dimensional grid. The following figure (Figure 3) represents the input connections in a self-organizing system. Every unit has its own input weight which is adaptive to implement self-organization. In the general case, the array of adaptive units receives its input signals through a relaying network which corresponds to the transformation that take place in the sensory pathways. The input signals are allowed to be mixed but the outputs from this array are always correlated. If three input signals, x1, x2, x3, are related according to some metric, then the output vectors, y1, y2, y3, from the relaying network must also retain the same order relating to some other metric.

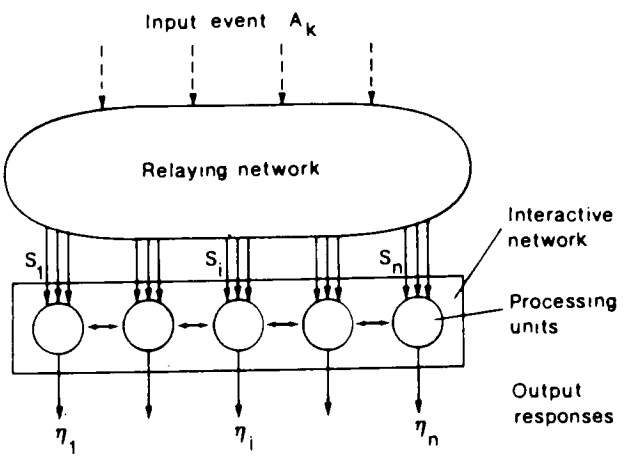


Figure 3: Input to a Self Organizing System, *(adapted from [Kohonen88a])*

If a physical system were to receive a set of input signals, these are transformed into output responses. These output responses are concentrated around some location in the output plane, and for different input vectors the location of this concentration is different. The localized responses may be thought to represent the center of an activity bubble (Figure 4).

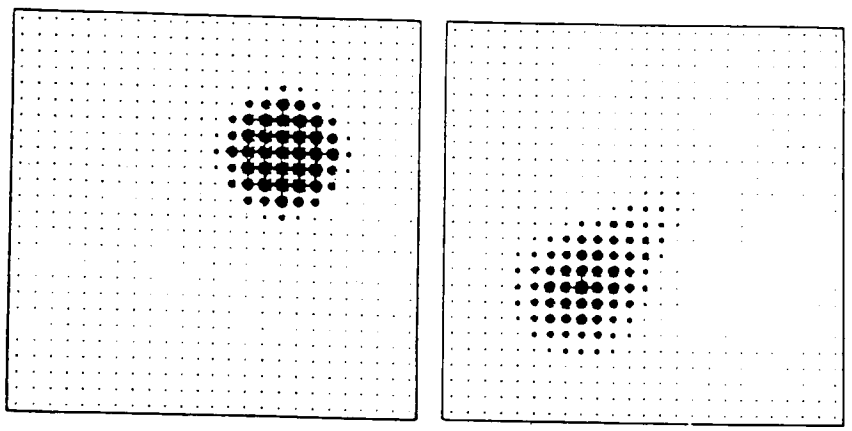


Figure 4: Activity Bubble, (*adapted from [Kohonen88]*)

The production of the two-dimensional mapping may be considered to be some sort of projection image of a higher dimensional distribution. If there are clusters or branches in the latter, they will also be projected as clusters and branches respectively (Figure 5). This mapping cannot be a parallel or orthogonal projection, but instead seeks an optimal orientation of the multi-dimensional input signal space for every part of the map. The input signal distribution is represented by the solid lines, and the dashed lines indicate the mapping onto a planar lattice of output nodes.

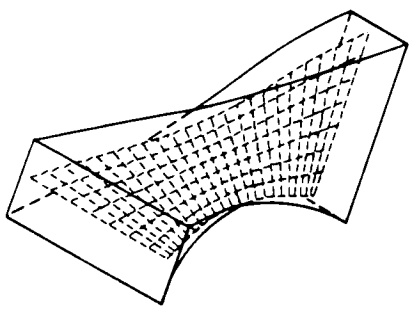


Figure 5: Projection of Higher Dimensional Distribution, (*adapted from [Kohonen82]*)

Self organization is also called *unsupervised learning* and uses unlabeled training data. No teaching is necessary, as in the backpropagation training algorithm which feeds back errors to the network to improve performance in future. Data is entered into the network and they form internal patterns or clusters. These clusters compress the amount of input data without loss of important information. This mode of learning is especially useful in domains such as speech and vision that require large amounts of data compression, reduced computation for classification, reduced space required for storage.

1.5 How Kohonen's neural net works

Self organized training in Kohonen's Self Organizing Feature maps uses an algorithm that creates a vector quantizer by adjusting weights from common input nodes to output nodes arranged in a two-dimensional grid (Figure 6). Output nodes are extensively interconnected with many local connections. During training, continuous value input vectors are presented sequentially in time without specifying the desired output. After a sufficient number of input vectors have been presented to the net, weights will specify cluster or vector centers that sample the input space so that the point density function of the vector tends to approximate the probability function of the input vectors [*Lippmann84*] A neighborhood needs to be defined around each node that decreases with time. The amount by which the weights are modified every time is also controlled by a gain term which decreases with time and tends towards zero.

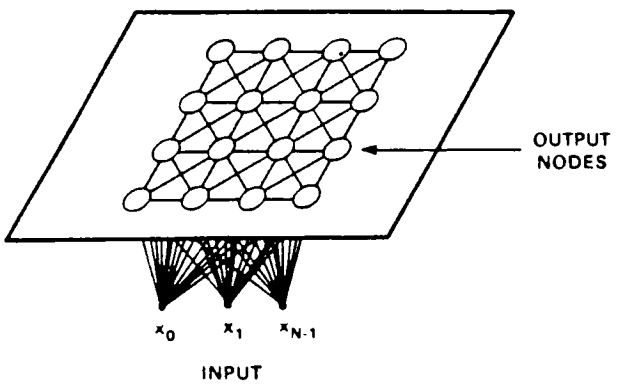


Figure 6: Two-dimensional array of output nodes used to form feature maps. Every input is connected to every output node via a connection weight, (*adapted from [Lippmann87]*)

Net training begins with setting random values for the weights from all input to all output nodes. Each input is presented to the net and the distances between the input and all the output nodes are computed. The output node with the least distance (i.e., greatest response) is then selected and all the weights to that node and all nodes in its neighborhood are updated. This makes these nodes more responsive to the current input. This process is repeated for further inputs until the weights converge and are fixed. Output layers in the end will be thus ordered in a natural manner. This may be of importance in complex systems with many layers of processing because it can reduce lengths of inter-layer connections.

Unsupervised training with unlabeled training data can often substantially reduce the amount of supervised training required. Abundant unlabeled training data is frequently available for many speech and image classification problems, but seldom is labeled data available. What is done is to use unlabeled data and feed it to a Kohonen map to produce a feature map and use information from the feature map as input to a Backprop, Hidden Markov Model or similar supervised classifier. This is useful because the Self Organizing Feature maps can perform very well and converge more quickly than other methods.

1.6 A Two-dimensional Self-organizing System

To illustrate the self-organizing mechanism, consider the units to be arranged in a rectangular planar configuration as shown below (Figure 7).

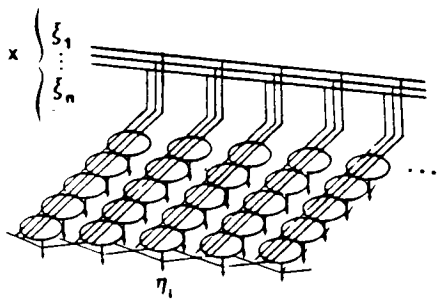


Figure 7: Two-Dimensional Self-organizing System, (*adapted from [Kohonen82]*)

Let each unit receive the same scalar signals $\xi_1, \xi_2, \xi_3, \ldots, \xi_n \in R$. Unit i, evaluates a function of these and in the simplest case the function is linear:

$$\eta_i = \sum_{j=1}^{n} \mu_{ij} \xi_j$$

where $\mu_{ij} \in R$ are the parameters which will be modified through the learning. Every ordered set $\{\mu_{i1}, \mu_{i2}, \mu_{i3}, \ldots, \mu_{in}\}$ may be regarded as a kind of image to be matched or compared against a corresponding ordered set $\{\xi_1, \xi_2, \xi_3 \ldots\ldots \xi_n \in R\}$. The aim is to devise adaptive processes in which all parameters converge to such values that every unit becomes specifically matched to, or sensitive to a particular domain of input signals in a some particular order. The location of the maximum output response may be considered as naming the "best match" between the vectors

$$x = \{\xi_1, \xi_2, \xi_3, \ldots, \xi_n\} \quad \text{and} \quad m_i = \{\mu_{i1}, \mu_{i2}, \mu_{i3}, \ldots, \mu_{in}\}.$$

The function described above is **correlation**; another commonly used matching criterion is the **Euclidean distance** between vectors. If the best match is defined at being at **unit c**, then **c** can be determined by the computation

$$\| x - m_c \| = \min_i \| x - m_i \|$$

A major advantage of this approach is that it allows the use of a simple mathematical technique. For the processing of the weights in the array we use a parameter $\alpha(t)$ that decreases with time and tends towards zero to guarantee convergence to a unique limit. The adaptation equations are of the form

$$d\mu_{ij}/dt = \alpha(t) \{ \eta_i(t) \cdot \xi_j(t) - \gamma[\eta_i(t)] \cdot \mu_{ij}(t) \}$$

The process that creates the clustering has to be taken into account. Around the maximally responding **unit c** a topological neighborhood N_c is defined such that all units which lie within a certain radius from the winning **unit c** are included in the radius.

Then for all units inside N_c one can have $\eta_i(t) = 1$;
and for all units outside N_c $\eta_i(t) = 0$;

If we further assume that $\gamma(0) - \gamma(1) = 1$, then the above equation can be reduced to the simple form

$$d\mu_{ij}/dt = \alpha(t)\{\xi_j(t) - \mu_{ij}(t)\} \quad \text{for } i \in N_c$$
$$d\mu_{ij}/dt = 0 \quad \text{otherwise}$$

The complete computational algorithm is expressed in discrete-time formalism in vector form as the following set of equations, where t_k stands for the discrete-time index.

Find the Best Match:

$$\| x(t_k) - m_c(t_k) \| = \min_i \{\| x(t_k) - m_i(t_k) \|\}$$

Update the weights in the neighborhood:

$$m_i(t_k + 1) = m_i(t_k) + \alpha(t_k)[x(t_k) - m_i(t_k)] \quad \text{for all } i \in N_c,$$

$$m_i(t_k + 1) = m_i(t_k) \quad \text{otherwise}$$

α is the gain sequence that is a slowly decreasing function of time. α may be taken to satisfy the rules of stochastic approximation, i.e., square summable:

$$\sum_{s=0}^{\infty} \alpha(s) = \infty, \quad \sum_{s=0}^{\infty} \alpha(s)^2 < \infty; \quad [\text{Example: } \alpha(s) = 1/s]$$

The problem of selecting a gain sequence is very subtle. In practice it is found that $\alpha(t_k)$ can be chosen as a reciprocal linearly decreasing function of time whereby the processing stops when $\alpha(t_k) = 0$. The proper choice for the values of the gain

function and the neighborhood can be presently obtained only from experience. This is the simplest representation of the algorithm. There are a number of variations that can be made to change the algorithm. Some of these have been dealt with and explained in the later parts of this thesis.

2.0 Speech and Vowel Overview

2.1 Phonetics and Phonemics

Phonetics seeks to provide an exact and unambiguous description of every known speech sound. It may be considered to be the study of speech sounds independent of any particular language. Phonemics on the other hand, is the study of speech sounds as they are perceived and thought of by speakers of a particular language.

Articulatory phonetics deals with the study of the production of speech sounds with emphasis on anatomical detail. Acoustic phonetics deals with the observable, measurable characteristics in the waveforms of speech sounds and especially those that enable them to be distinguished from one another. An important goal here is to determine the relation of these acoustical characteristics to the positions of the speech organs. Acoustic phonetics provides the framework for speech recognition and synthesis using electronic hardware. A phoneme is the basic unit in phonemics in much the same way as a phone is an individual sound in phonetics.

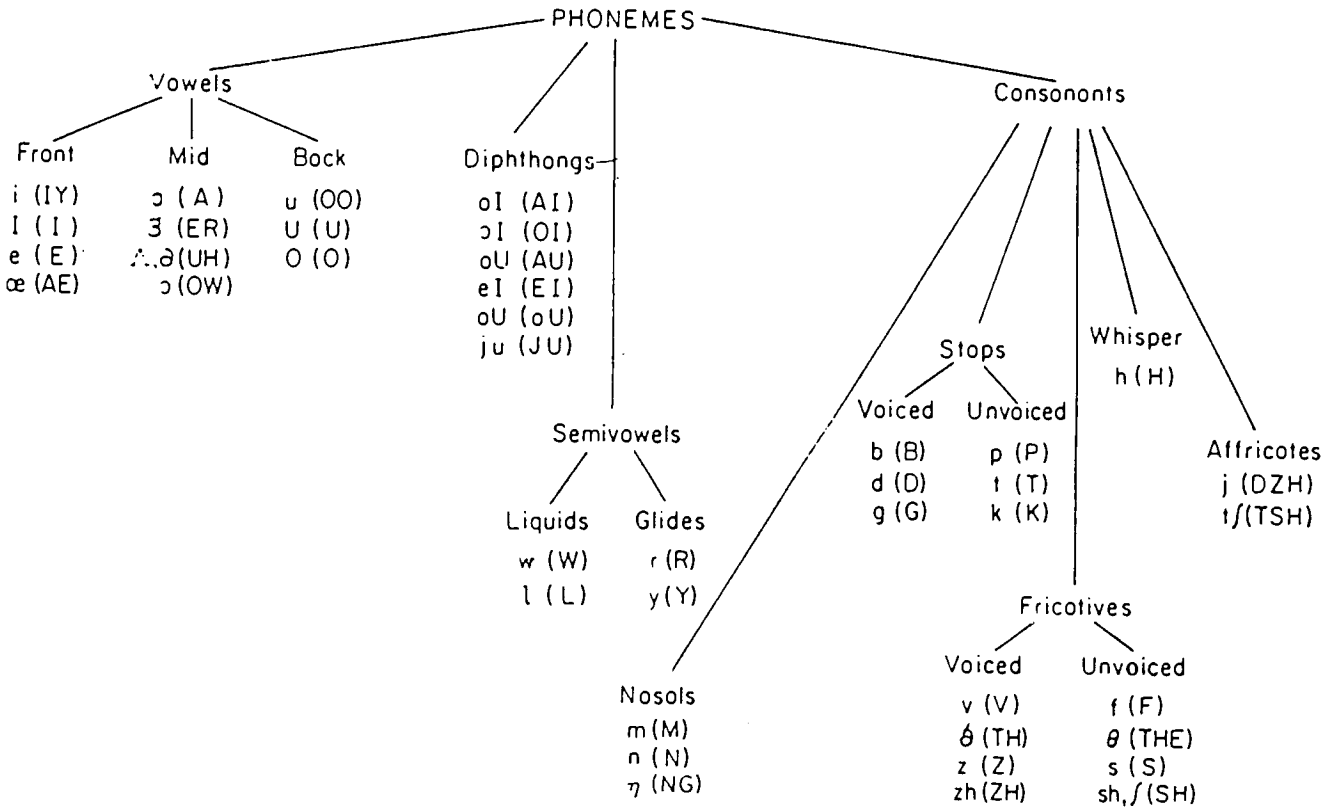


Figure 8: Phonemes in American English, (*adapted from [Rabiner87]*)

Figure 8 shows the phonemes that are present in the American English Language. There are approximately 42 basic sounds. Vowels constitute one of the classes of phonemes. The primary focus of this thesis is on the classification of vowels.

Each phoneme possesses certain characteristics that makes it distinguishable from others and these can be used for classification purposes. The key to recognizing phonemes is to be able to identify these traits, segment speech and extract the phonemes from the speech signal. Boundaries are difficult to impose upon the speech signal for segmenting purposes. This is because the distribution of spectral samples of different phonemes overlap, even in clear speech from the same speaker. The same phonemes spoken by different persons can be confusing too. As a result, accurate classification of phonemes in a speaker-independent environment is very difficult.

The production of a phone will include some articulatory features left over from the previous phone and some anticipation of features in a subsequent phone. This overlapping of phonetic features from phone to phone is called **coarticulation**. This phenomenon occurs when the characteristics of one phoneme changes as a result of the surrounding phonemes. Vowels are easily distinguished from other phonemes by their formant frequencies. The first three frequency locations of the formants, referred to as F1, F2, F3, are commonly used for classification purposes. Vowels are visually identifiable in spectrograms by their formant frequencies. They appear as dark bands of energy. The frequency locations of the first two formants are closely tied to the shape of the vocal tract as the lips, tongue, pharynx and jaw move to articulate the vowels and consonants [*Picket80*].

The context under which words are uttered also has an important part in influencing the acoustic properties of the phonemes. Hence, it may be said that the speaker, coarticulation, and context of the uttered word are different aspects that complicate the identification of phonemes. Although there are a relatively small number of phonemes, the above factors cause their identification to be a difficult task and requires a complete understanding of the acoustic properties and the interaction between phonemes. For a speech signal to be used as input for a neural net, it has to be converted to a form that is acceptable to the system. The preprocessing that is done carries out the conversion of the analog speech signal into its digital form. Once the processing has been done, analytical techniques are

used to extract features that uniquely identify the signal and this data is then used for classification.

2.2 Vowels

2.2.1 Source Filter Theory

Speech production is viewed as a process which involves a sound source that excites the resonances of the oral and the nasal tract. This model that is used extensively in the analysis and synthesis of speech is referred to as the ***source filter theory.*** According to Minifie, [*Minifie73*], " *the properties of vowels may be described in terms of the acoustical properties of sound source and the modification of that source by the acoustical filtering that takes place within the vocal tract*". The source filter theory of speech production is dependent on **three** factors: (a) a source of sound energy, (b) a vibrating body, and (c) a resonator. The accompanying figure (Figure 9) illustrates the concept of the source filter theory.

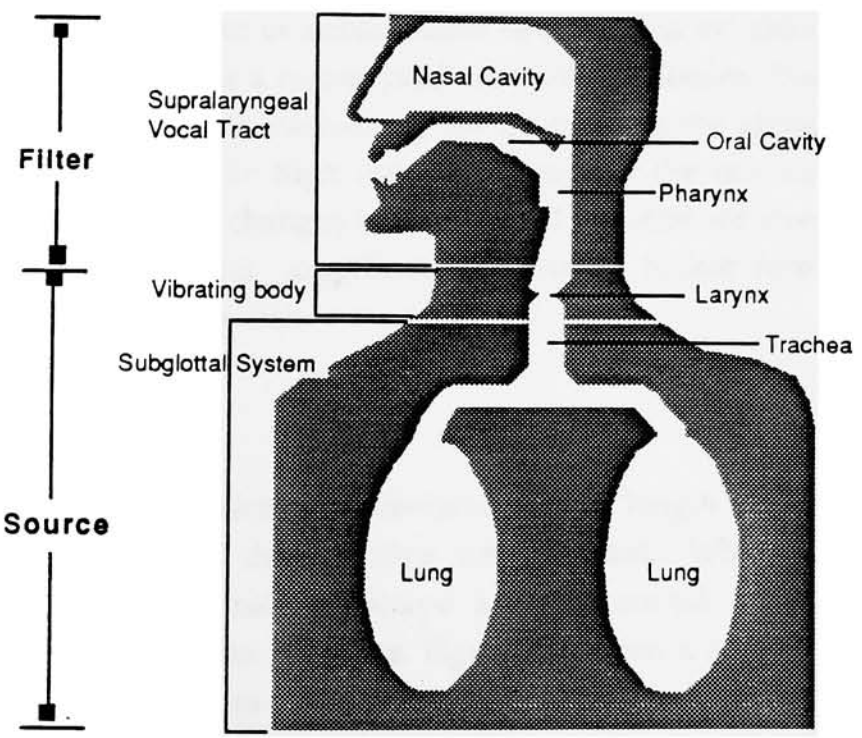


Figure 9: Source Filter Theory, (*adapted from [Stam90]*)

2.2.1.1 The Source

Vowel sounds are produced by repeated acoustic excitation of the vocal tract air column with a series of glottal pulses. Air flows up from the lungs through the vocal folds, into the pharyngeal and oral tracts and then out of the mouth. This action occurs in pulses and "*acts as an outward airflow disturbance that is propagated back toward the vocal fold surface*" [*Picket80*]. The vocal folds act like the base of a bottle and reflect the pulse back upward again in a periodic manner. These folds can be opened for normal breathing, left completely closed or left partially open. The opening between the vocal folds is called the glottis. During voicing, the glottis is open partially, and the air is allowed to escape from the lungs; this output of air causes the folds to vibrate. The degree of pressure applied to the vocal folds controls the frequency of the vibrations [*Wilder75*]. The glottal tension and the subglottal air pressure affect the characteristics of the sound source especially the pulsing rate. By increasing the subglottal air pressure, the rate of repetition of airflow increases, which means that the pitch increases. The pitch is the fundamental frequency associated with the sound generated. Pitch is also affected by the amount of stress placed upon certain syllables. In American English stressed syllables have a higher pitch than other syllables. Due to the increase in the subglottal pressure, the intensity of the sound from the glottal source increases and consequently raises the high frequency range of the spectrum relative to the low frequency range. The changes in tension and pressure for stressed vowels produce a higher second formant amplitude and overall higher formant frequency values than for unstressed vowels.

2.2.1.2 The Filter

The filter's shape, which is determined by the length of the pharyngeal and oral passages, determines the resulting vowel sound. When a vowel is uttered, the vocal tract takes a tubular shape and the sound produced contains certain resonating patterns. As is shown Figure 10, when a specific filter response of the vocal tract is applied to the glottal sound spectrum, a particular vowel spectrum is produced based on the filter. The important thing to understand is that, "*the glottal source sound spectrum is the same for all the vowels and this spectrum is changed by the filtering of the vocal tract to produce sounds that have different patterns*"

[*Picket80*]. When the vocal tract assumes other shapes, different patterns of vowel sounds are correspondingly produced.

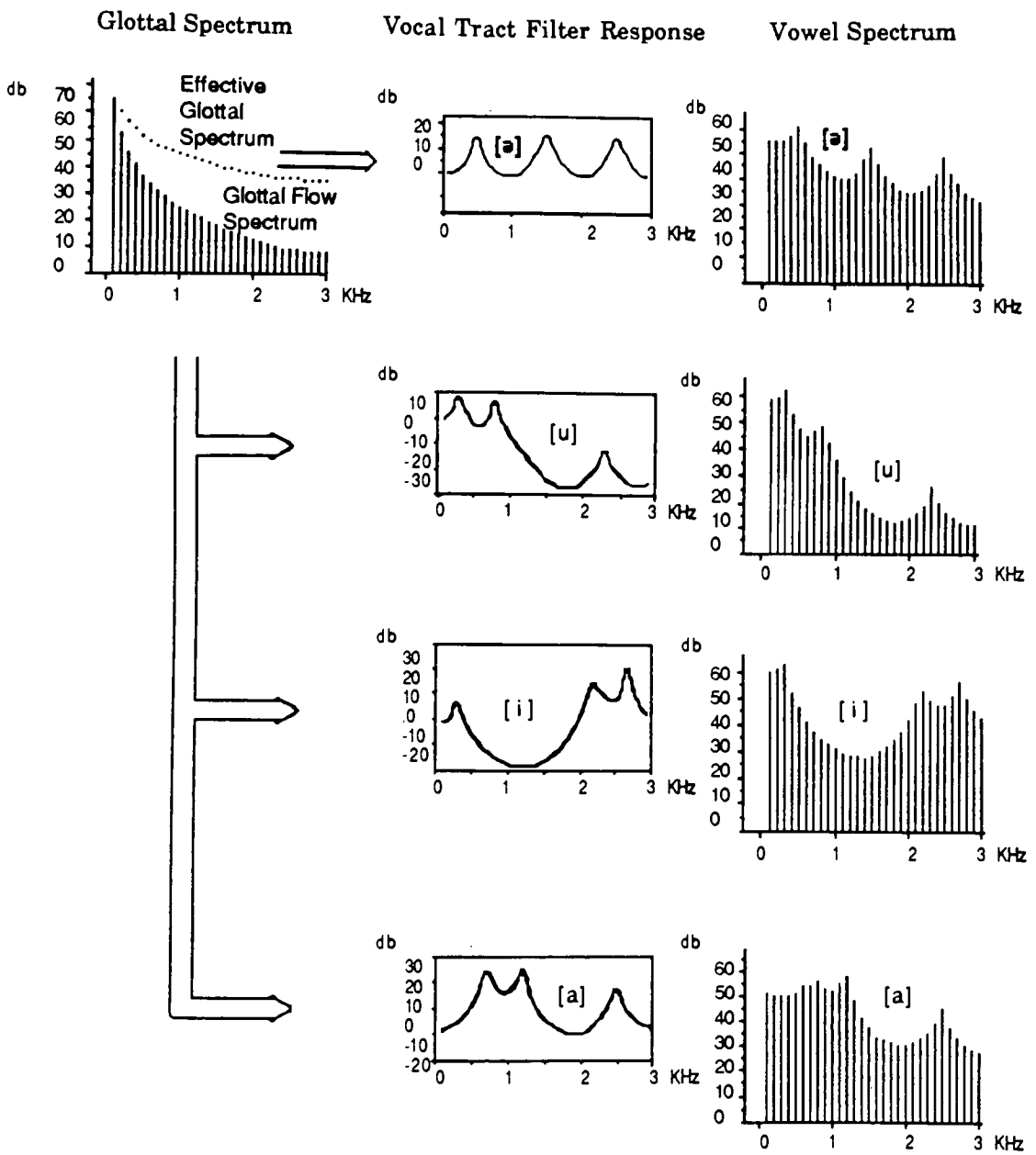


Figure 10: Filter Responses of the vocal tract, (*adapted from [Stam90]*)

The length of the vocal tract in the average male is 17.5 cm from the glottis to the lips. The voicing that occurs during the production of sound has many natural frequencies associated with it. These are the frequencies at which the transfer function associates with the tube would be at a maximum. These frequencies are referred to as formants, and correspond to the dark bands that appear in the spectrogram. *"As the shape and size of the resonance cavities are altered during speech production, formant frequencies are also changed so that every configuration of the vocal tract has its own characteristic formant frequencies"* [*Wilder87*].

2.2.2 Vowel Production

Vowels are produced by modifying the source by the acoustical filter. This is done by the combined effect of the tongue, lips, jaw and takes place within the vocal tract. Vowels are classified as being of three types: **front, middle and back vowels**. It is the location of the tongue that enables us to classify different vowel sounds based upon the point of constriction. As seen in Figure 11, the front vowels have the point of maximum constriction of the tongue located near the alveolar ridge. For the back vowels, the highest point of the tongue is near the velum. The remaining middle vowels are produced when the tongue is in a more central position. The vowel triangle illustrates the location of the different vowels based on these positions. The degree of constriction of the tongue is determined by the height of the tongue. High vowels such as /iy/ and /uw/ have a highly constricted tongue during the utterance whereas low vowel such as /ae/ and /aa/ do not.

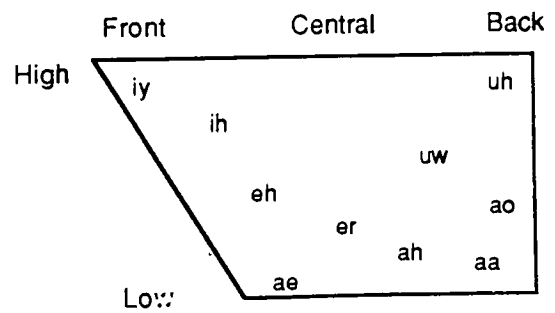


Figure 11: Vowel Triangle, (*adapted from [Parsons87]*)

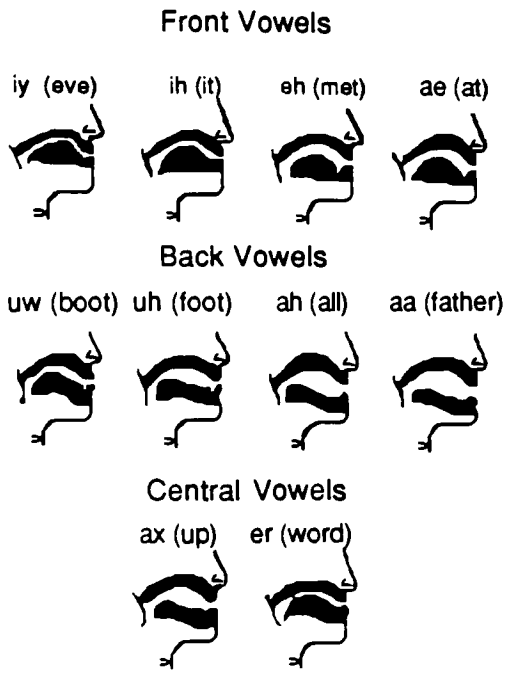


Figure 12: Vowel production, *(adapted from [Stam90])*

2.2.3 Nasalization in Vowels

"*Vowels sounds in English are spoken with the velum raised against the walls and back of the pharynx to shut off the nasal passages completely from the pharynx and the oral tract*" [*Picket80*]. Leaving the entrance to the nasals cavities open by lowering the velum can introduce a nasalized sound to the uttered vowel. The addition of the nasal branches to the vocal tract creates a longer and larger resonator. This affects the formants by lowering the frequencies. Nasalization also occurs in vowels that are adjacent to nasal consonants. This causes an attenuation of upper formants relative to those of the neighboring vowels. The damping of the resonances is partially due to the a broader band frequency response set up in the elongated tract. Another reason is that the sound is absorbed by the soft walls of the convolutions of the nasal cavities. "*This causes a reduction in F1 amplitude and insertion of anti-resonances and extra formants in the transmission of the pharyngeal and oral tracts, thus altering the normal vowel spectrum*" [*Picket80*].

2.2.4 Vowel Acoustics

The frequency locations of the formants are closely related to the shape of the vocal tract as the filter is transformed to articulate vowels. The most thorough study of vowel formant frequency patterns was conducted by Peterson and Barney [*Peterson52*]. The study was conducted in two parts. The first part investigated the relationship between the phoneme intended by a speaker and that identified by the listener. Peterson and Barney made a list of 10 monosyllabic words each beginning with an /h/ and ending with a /d/ and differing from one another only in the vowel. The words that were used were **heed, hid, head, had, hod, hawed, hood, who'd, hud** and **heard.**

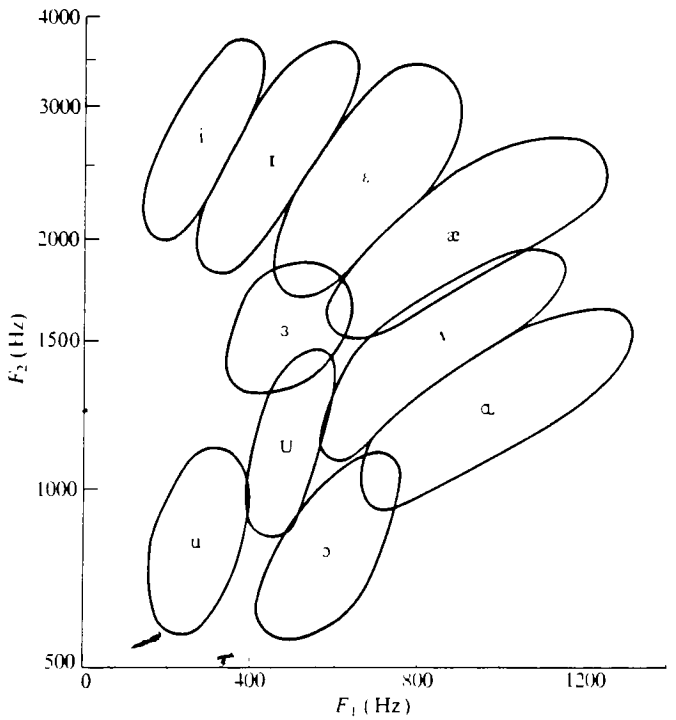


Figure 13: F1 plotted against F2, (*adapted from [Parsons87]*)

A group of 76 speakers, that included 33 men, 28 women and 15 children each recorded two sets of the ten vowels to produce a total of 1520 recorded words. These words were randomized and presented to a group of 70 listeners, 32 of whom were

from the original set of 76 speakers. Listeners were asked to classify each word into one of the ten possible categories based on the vowel sound they heard. The overall classification accuracy of the listeners was 94.4%. This proved that humans listeners were highly capable of identifying vowel sounds.

The second part of their study involved classification of the words that were recorded by means of a spectrograph. A measurement of both the frequency and amplitude of the formants for the 20 words recorded by each of the 76 speakers was made. These measurements were made during the steady part of the vowel that following the influence of the /h/ and preceding the influence of the /d/ [*Peterson52*]. They plotted F1 against F2, F1 and F2 being the first and second formant frequencies. Figure 13 shows the graph that was produced by the study. It can be seen that the ten vowels are not totally separable.

It is not quite clear how human listeners are capable of successfully classifying vowels with such a low rate of error (approximately 5%) . The ovals or hoops in the figure represents the data points for each category. Each of these encompass about 90% of the data points that belong to the same phonetic category. The vowels overlap with the neighboring ones and not all vowels are classified as belonging to their correct category. This finding, and the high accuracy in human recognition, suggests that human listeners must use more information than what is present in the first two formant frequencies.

2.2.5 Coarticulation Effects in Vowels

The Peterson and Barney data showed a relatively strong relationship between vowel identity and formant frequency patterns. But it should be realized that all the vowels were produced in single words in the same phonetic environment. As has been discussed above, the speaking rate, linguistic stress, phonetic environment also have strong effects on the formant frequency patterns of the vowels. Each vowel in the case of the Peterson and Barney Data is characterized by its target formant values within an acoustic "vowel space".

"*Thus for example, if a vocal tract configuration for a vowel is realized by instructing the tongue mass to move in a particular direction, the actual displacement of the mass will lag behind the instruction and will not approach the*

specified position unlike after some time has elapsed. During the production of the syllable, therefore when forces maneuver from a consonant configuration to a vowel configuration and back, the displacement that results from this combination of forces may fall short of the displacement associated with the ideal target vowel configuration" [*Stevens63*].

Stevens and House [*Stevens63*] conducted a study of the perturbation effects of vowels in 14 consonantal contexts. Using only three male speakers, they were able to show that systematic shifts in the vowel formant frequencies depend upon the particular vowel, the voicing characteristics and the manner and place of production of adjacent consonants. Figure 14 shows the effects of coarticulation due to consonantal context. In this figure, the values of F1 and F2 for eight different vowels are averaged for the 3 speakers and then are plotted to demonstrate the effect. The consonantal effects included are velars, postdentals, labials, and the null environment (an average of the /h/ vowel/d/ context and the isolated vowel) .

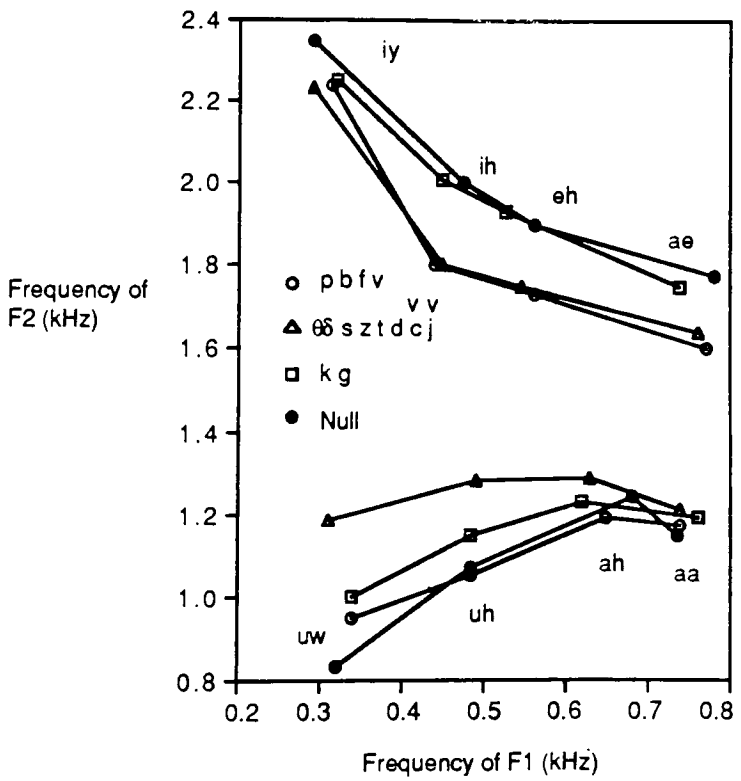


Figure 14: Effects of place of articulation on consonantal context, *(adapted from [Steven63])*

These are represented by squares, triangles, hollow circles, and filled circles respectively. The average shift in F1 is small for different places of articulation of the consonant. The changes for F2 are, on the other hand, are clearly apparent. The amount of deviation from the null case for F2 depends on the particular vowel being examined. The amount of downward F2 movement is least noticeable for velar consonants spoken with front vowels and for labials spoken with back vowels. The greatest variation in F2 occurs when front vowels are joined with the labials and postdentals. Deviation of back vowels also occurs when the back vowels are joined with postdentals.

2.2.6 Feature Selection

Input data that is fed to any classifier or recognizer is a mix of relevant and irrelevant data. Feature selection is the process of removing as much irrelevant information as possible and representing relevant data in a compact and meaningful form. Any utterance contains information about the words being spoken and also about the identity of the speaker. It is very important to select the type of features that are going to be used in the classifier. Features being used should ideally meet the following criteria [*Parsons87*]:

* Vary widely from one class to the other
* Insensitive to extraneous variables such as the text, context, health, emotion etc.
* Stable over long periods of time
* Occur frequently
* Easy to measure
* Not correlated with other features

It is very difficult to find features that meet all of these requirements. Feature selection is often based on the understanding of the anatomy, phonetics and other aspects that govern the production of speech.

Features have to be evaluated before they are selected to see how well they separate the different classes. There are different techniques that are used to evaluate single features and combinations of features. There also exist different measures that indicate separability of the features. All these serve to help us decide as to which

out of a large number of possible features to include and which to ignore. Three approaches that are commonly used are:

(a) **Knock-out** and **Add-on** algorithms
(b) **Dynamic Programming** and
(c) **Transformations.**

The **knock-out** algorithm [*Sambur75*] states that a feature may be dropped if its omission does not degrade the classification accuracy. If we have N features, then test N-1 subsets and see how they degrade performance by omitting each of the features in turn. Remove the one feature that degrades the performance the least. Repeat this for the remaining N-1 features, then for the remaining N-2 and so on. The order of knocking out gives the ranking of the features. The least useful features are the first to go. The reverse of this process is the add-on algorithm [*Goldstein76*]. Each of the N features are considered in isolation, and the best feature forms the nucleus of the final set. All pairs comprising the nucleus and one feature are evaluated, and the feature whose addition results in maximum improvement of the performance is added to the nucleus. This process is repeated until all features have been considered or until the desired performance has been obtained. The only problem with the Knock-out and the Add-on techniques is the number of tests that have to be made.

Knock-out requires a total of [(N(N-1) -K)K-1] /2 evaluations for a subset of K features.

Add-on requires lesser evaluations than the Knock-out. It requires K(2N+1-K)/2 evaluations.

Correlation effects among features plays an important role in these methods of feature selection.

Dynamic programming is another technique that is used for feature selection. The above two techniques do not guarantee that the resulting subset will be the best possible combination of features. This technique ensures finding an optimum subset of features. It is similar to the add-on technique except that in the case of the add-on algorithm, we have a single subset that is growing; for dynamic

programming we have many subsets that are growing in parallel, and ultimately the best is selected.

The third technique that is used is called **transformations**. This refers to rotations or modifications of the coordinates of the feature space. The general object of these transformations is to identify those features that help in separating the classes. Transformations are of two types, the first wherein we uncorrelate the features and the second wherein we try to maximize separability. Unfortunately the transformations are often computationally expensive, and the resulting components are many times not meaningful.

2.2.7 Clustering

Features in many cases may separate classes well enough that the classes occupy nearly non-overlapping regions in the feature space. In such cases, it is possible to allow the classifier to group the training data into categories without supervision. This gives rise to the process known as clustering. A number of clustering algorithms have been developed, prominent among which is the **K-Means clustering algorithm**. The formation of clusters using this algorithm closely resembles the formation of Feature Maps using the Self-Organizing process.

In the K-Means clustering process, we initially set the means of the k clusters by choosing the first k data points. Then every new data point x_i is assigned to the cluster whose mean is nearest to x_i. After all data points have been assigned, the cluster means are recomputed. This process is repeated until no change in the means occurs. This is one of the most efficient and popular clustering technique. A weakness in this algorithm is that the performance can be affected by the choice of the initial clustering. This initial choice may cause the algorithm to converge to a local optimum instead of a global optimum.

In conclusion, it may be said that according to the source filter theory, vowel sounds are produced by the shape of the vocal tract and the amount of tension applied to the vocal folds. Different vowels are produced by changing the source of the sound energy, the vibrating body, the resonator and a combination of these. Classification of vowels requires finding an appropriate set of features that adequately characterizes the vowel region. The important characteristics involved in the

recognition of vowels are the formant transitions between phonemes, the location of the first three formant frequencies and the overall shape of the spectrum. Feature selection is an important aspect that helps classify the input effectively.

3.0 Vector Quantization

3.1 Acoustic Preprocessing

Physiological research on hearing has revealed that many details of the acoustic signal may or may not be significant to artificial speech recognition. The main operation carried out in the ear in human hearing is acoustic preprocessing in the form of frequency analysis. This spectral decomposition of the speech signals is transmitted through to the brain via the auditory nerves. Each peak of the wave pressure gives rise to separate bursts of neural impulses and in this way some time domain information is also transmitted to the brain. Phase information is conveyed by synchronization of the neural impulses.

Acoustic preprocessing usually includes the following:

* Noise cancellation with a suitable microphone
* Pre-amplification with a low-pass filter
* Analog to digital conversion at a suitable sampling frequency.
* Spectral decomposition using a Fast Fourier Transform or Linear Predictive Coding to encode the speech signal, with as suitable window size. For the Kohonen Self Organizing Feature Map the FFT is more desirable as it reflects clustering properties better than a parametric code like a LPC.
* Formation of the vector (Vector quantization), grouping and subtraction.
* Normalization of the vector into constant length.

3.2 Vector quantization

Vector quantization is a coding scheme that is used to encode given input data in an economical fashion. The simplest form of vector quantization is with the use of a codebook. A codebook is a collection of messages each of which is indexed by a k-bit number. The transmitter selects a message from the codebook and transmits the k-bit address. The receiver enters its copy of the codebook at this address and recovers the message [*Parsons87*]. Each message in the codebook may be much more than a single sample. The entries in the codebook may be a sequence of samples, i.e. a

vector. The longer the vectors, the better is the quantization. The only key is the selection of a well chosen codebook.

3.3 Speech and Vector Quantization

Vector quantization of speech waveform involves an *encoder, a decoder and a codebook*. The figure (Figure 15) below illustrates this concept. The codebook is a lookup table with a k-bit address and 2^k entries. Each entry in the table is a vector of samples. Both the coder and the decoder have copies of the codebook.

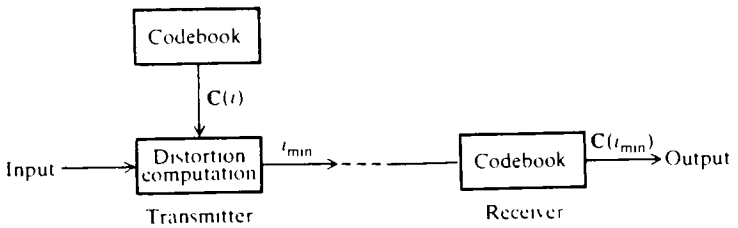


Figure 15: Vector Quantization Scheme in Speech, *(adapted from [Parsons87])*

The encoder looks at each incoming vector of samples and selects the codeword that is the best match to the incoming vector. The measure of the error in representing the vector is termed the **distortion measure**. The Euclidean measure is a frequently used measure although any positive definite quadratic measure is sufficient. The encoder transmits that codeword for which the distortion measure is the minimum. The decoder receives its input and presents the actual vector from its codebook as the output. The only problems with vector quantization are the amount of time required for searching a large codebook and the requirement of an accurate and meaningful method of determining the distortion measure (low distortion should correspond to a good sounding output) [*Parsons87*]. Each of the vectors that have been mentioned are patterns and have elements that are measured values of features. A feature may best be described as some measurable characteristic of the input which has been found to useful in recognition. If the pattern takes the form of a set of time function-values for each feature recorded over the length of the utterance rather than at particular points, it is termed a

template. Vector quantization, in short, is a technique that is used for data compression of signal information and for separation of signals into categories.

3.4 Self-organization and Vector Quantization

The various spectral values that are obtained from the FFT can be regarded as being an **N** dimensional real vector, where N is the number of channels formed from the FFT, in a Euclidean space. We can think of the spectra of the different phonemes of speech occupying different regions in **N** space and use a multi-dimensional discrimination to identify them. Problems in this approach are (a) the overlap of the various phonemic classes and (b) coarticulation effects. Hence the best alternative is then to divide the space with optimal borders, relative to which the rate of mis-classification is reduced. It has been observed that analytical definitions of such borders are far from trivial whereas neural networks can define them very effectively *[Kohonen88].*

A concept useful for the illustration of the vector space methods is called *Voronoi Tessellation* (Figure 16). To illustrate this, we can consider that the dissimilarity of two or more spectra to be expressed in terms of their vector differences in an N-dimensional Euclidean space. The figure shows a 2-D space with a finite number of reference vectors. The space is partitioned into regions, bordered by lines, such that each partition contains a reference vector that is the nearest neighbor to any vector within the same partition. The lines or the **midlands** [*Kohonen88*] constitute the Voronoi Tessellation that defines a set of discrimination or decision surfaces. This representation represents one kind of vector quantization, which means quantization of the vector space into discrete regions. One or more neighboring reference vectors can be made to define a category in the vector space as the union of their respective partitions. An approach to this is to use samples or prototypes of earlier vectors as such for reference vectors. For a new or unknown vector a small number of its nearest neighbors are sought, and majority voting is applied to them to determine classification. A problem here is that for good accuracy a large number of reference vectors are needed.

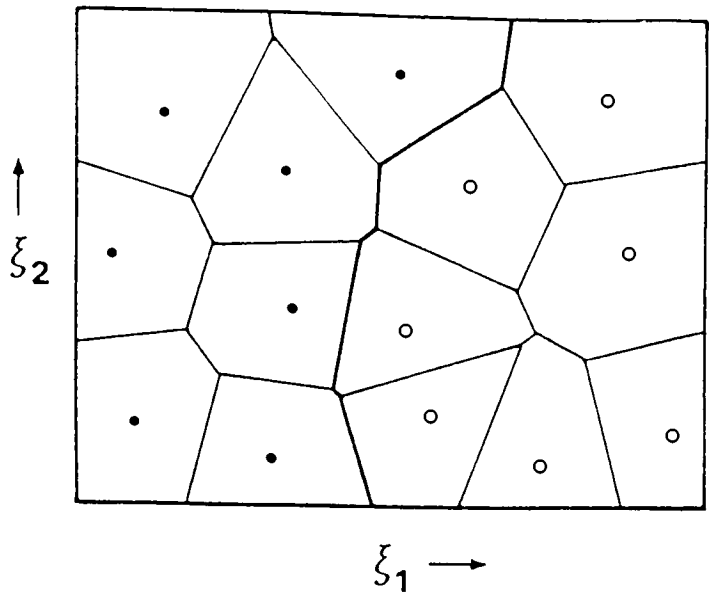


Figure 16: Voronoi Tessellation, *(adapted from [Kohonen88])*

If the input signal is presented to a fixed number of neurons in parallel, and these neurons have a template against which the degree of matching of the input spectrum is determined, the different neurons compete and the neuron with the highest match is considered to be the winner. The input spectrum would be assigned to the winner in the same way an arbitrary vector would be assigned to the closest reference vector and classified according to it as in the Voronoi Tessellation. The neural nets can have templates formed adaptively and perform comparison in parallel, so that the neuron whose template matches best with the input automatically gives an active response to it. The self-organizing process defines reference vectors for the neurons such that their Voronoi tessellation sets near-optimal decision borders between the classes, i.e, the fraction of input vectors falling on the wrong side of the borders is minimized. As has been explained, the self-organizing maps have a bearing on vector quantization. A characteristic that makes them resemble biological maps is the spatial order of their responses, which is formed in the learning process.

3.5 Learning in the Self-organizing Feature Map

In a time continuous process, the weights vectors attain asymptotic values, which then define a vector quantization of the input signal space, and thus, a classification of all its vectors. In practice, the same vector quantization can be computed much

more quickly using a simpler numerical algorithm. The bubble (presented earlier in Figure 4) is equivalent to a neighborhood set N_c of all those network units that lie within a certain radius from a certain **unit c**. The size of the bubble depends on the interaction parameters, and so the radius of the bubble is controllable, eventually being definable as a function of time. For good self-organizing results, it has been found that the radius of the bubble should decrease monotonically.

A shortcut learning algorithm is used to control the radius of the bubble, which is as follows:

* Start with initial values $m_i = m_i(0)$.
* For t = 0,1,2,3,4, . . .compute

    * Determine the center of the bubble(c):

$$|| x(t) - m_c(t) || = || \min_i \{|| x(t) - m_i(t) ||\}$$

    * Update the weight vectors:

$$m_i(t+1) = m_i(t) + \alpha(t)\,(x(t) - m_i(t)) \quad \text{for } i \in N_c$$
$$m_i(t+1) = m_i(t) \quad \text{for all other indices i}$$

It is assumed that $\alpha = \alpha(t)$ and $N_c = N_c(T)$ are empirical functions of time. The asymptotic values of the m_i define the vector quantization. If N_c contained the index i only, the two equations would resemble the classical vector quantization method called K-means clustering. This method is more accurate as the corrections are made over a wider, dynamically defined neighborhood set, or bubble N_c, so that an ordered mapping is obtained.

3.6 Phonotopic Mapping

Mapping can be used as a principle for the visualization of speech signals. A two-dimensional map formed by self-organization may be used to display the similarity relationships of phonological units. This is a very general map as it takes into account the complete spectra. This technique of mapping extracts a few (usually two) of the most important features of the pattern space and displays the pattern vectors in such a low coordinate system. The selected features are the same for the

whole pattern space, but optimal feature dimensions for every region of the pattern space are determined dynamically. Hence it may be said that it is that it is the local topology, not the global topology, that is preserved in the map. This type of mapping mechanism may be used for the visualization of continuous speech and for phonetic labeling and segmentation of speech waveforms. These representations are called phonotopic maps. These mappings are explained by assuming a two dimensional array of nodes or units which are arranged in a hexagonal or rectangular planar lattice (Figure 7).

The simplest type of phonotopic maps formed by self-organization is called the phoneme-map (Figure 17). The map shown here is for the phonemes in the Finnish language. The network when employed for this type of mapping has no segmented or labeled speech. All features that are present contribute to the organization as is seen at the end of the training. Superficially, the network seems to have only one layer of neurons. Due to the lateral interaction in the network, the topology has an even more complicated effect than Boltzmann machines or the backprop networks. Any neuron in the network has the ability to create an internal representation of the input information in the same way as the hidden units in the backprop networks do. An interesting phenomenon that is observed when the speech spectra is presented to the various nodes is the sensitivity that neurons acquire to different spectra of phonemes and their variations in a two-dimensional order, as teaching is done only by the spectral samples and not by the phonemes themselves. The reason is that the input spectra tend to cluster around phonemes, and the self-organization finds these clusters. The map can be calibrated then using spectra of known phonemes. If a new or unknown spectrum is presented to the net, the neuron with the closest transmittance vector, say m_i, gives the response, and so the classification occurs in accordance with the Voronoi tessellation in which m_i acts as the reference vector. There could be two variations of phoneme maps if we have a two dimensional grid. As is discussed later in the implementation, the first variation shows at which processing unit each phonemic unit caused the maximum response. The second variation could be a map that is obtained by looking to which phoneme each unit became most sensitive and labelling it accordingly. These two have been implemented and can be visually be seen in the graphical user interface.

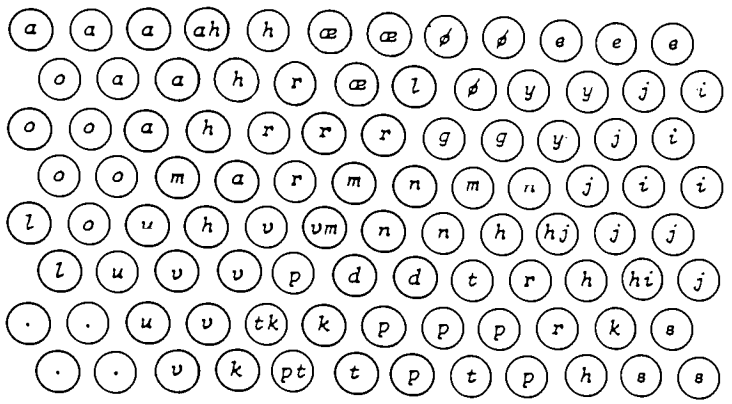


Figure 17: Phoneme Maps, *(adapted from [Kohonen88])*

4.0 Implementation

4.1 Functional Description

This thesis has three major components. The first is the implementation of Kohonen's Self-Organizing Feature Map. The second consists of the training and testing of the vowel data with this network, the analysis of the performance and comparison with other classification methods. The final component deals with the implementation of the graphical user interface using **Xview**, an X-Window system based toolkit that supports "**OPENWIN**". This chapter deals with the implementation of the neural network and the graphical user interface.

4.1.1 The Neural Network

The neural net has been designed in a modular manner so that future changes and modifications are easy, and it may be run in different environments with a minimum of change. It has also been designed in such a way that adding new GUI components to the existing one will be easy. The neural net takes for its input raw speech data in the form of spectral values or data in the form of extracted features from recorded utterances. The output from the neural net is a set of node activations for the inputs that were presented to it and a confusion matrix that describes the accuracy of the classification. The user has the option of training the net from the beginning, continuing training from an earlier run or testing the net for its performance based on its current training. This is made possible by periodically writing to a file the dimensions of the net, the weights for the connections and all other important parameters associated with the current state of the network.

The input for the network (when used without the graphical interface) is interactive. The program prompts for values of different parameters from the user, such as the net topology, input features and the data file that are essential for it. When the user chooses not to supply some of these parameters, reasonable default values are supplied, and the network starts to function.

The output in the training and the testing phases is in the form of tables that describe the classification of the various vowels as seen by the net. The tables are composed of a two dimensional grid with the same dimensions as the output layer of the network. A table is printed out for each of the vowel classes. Each table shows

the response caused at each output node by the vowels of a particular class. Since we have ten vowels whose classification is being studied, ten tables are printed out. Another table, which is the summary table, describes at which unit each of the phonemes elicited maximum response. A confusion matrix is also printed that describes the error rate in the classification at that time. To obtain the confusion matrix, the nodes in the neural net are labelled as belonging to a particular class based on the number of activations it has had. A node is said to belong to class "i" if it responded the maximum number of times to input belonging to class "i" from among all of its responses. All this information is taken in by the program that constitutes the graphical interface for drawing the different phonotopic maps on the screen.

4.1.2 The Graphical User Interface

The graphical user interface is made up of two parts. The first part comprises the frames that interacts with the user to obtain input to start up the neural net. The second part consists of taking the output produced during the first stage and drawing on the screen the different phonotopic maps describing the interrelationships between the vowels as the neural net organized it at that stage. The same functionality has been retained as in the previous stage where the user runs the neural net without this interface. The user here also is presented with the choice of starting a new training session, continuing from a previous run, testing the neural net, or changing default values associated with the parameters for the net.

During the first stage the user is presented with options that to choose from. Frames pop up that the user can use to fill in values to be used as the parameters for that run. Default values are provided in all cases. Once the net has started to run, the output from it is the same as described in the previous section. Every time the weights are written to a file, another file is created that contains information about the activations of the nodes for all the inputs besides information about the topology of the network. This is to facilitate consolidation at a later stage of all this information for observing the characteristics of the net using the graphical interface for a large amount of accumulated data all at once. The Graphical interface (phonotopic map display program) uses colors to indicate the different vowels. Each of the different classes has a color associated with it. There are six different

maps that are presented to the user in this part of the interface for each set of data that is available. An attempt has been made to provide the user with all the different possibilities that exist for the interpretation of the data. A detailed description of the features of the interface has been provided in the later sections of this thesis.

4.1.3 Data Files for the Neural Net

The data that has been used for the simulations of the neural net are the **Peterson** and **Barney** data and the **Hillenbrand** Data. Both these sets are made up of vowel data. The Peterson and Barney data is made up of 1520 vowel samples that have been extracted by hand from speech utterances. Each vowel sample has a total of eight features. They are the pitch, the first, second and third formant frequencies represented by F0, F1, F2 and F3 respectively, and the means of these for each speaker over the entire data set, represented by mF0, mF1, mF2 and mF3, respectively. There is also for each vowel sample two other fields in the data files. They are the serial number of the vowel from the original data file and its class. The vowel class for the ten vowels lie between zero and nine. The accompanying Table shows the ten vowels that were used for the training and testing. The table also shows their class numbers, their IPA symbol, an example of the vowel used in a word and the arpabet symbol for that vowel.

Vowel Number	Arpabet	Example	IPA Symbol
0	IY, i	HEED	i
1	IH, I	HID	I
2	EH, E	HEAD	ɛ
3	AE, @	HAD	æ
4	ER, R	HEARD	ɝ
5	AH, A	HUD	ʌ
6	AA, a	HOD	ɑ
7	AO, c	HAWED	ɔ
8	UH, U	HOOD	U
9	UW, u	WHO'D	u

Table 1: Vowels Used for training and testing the Network

The data has been normalized with the standard deviation of every feature made to be one. The order in which each vowel sample appears has been made random to prevent the occurrence of any bias during the training process. The data file with the eight features has been used to produce a smaller data file consisting of only the first four features. This data file is called f0_f3. One of the limitations that the program imposes on the user regarding the data file that is being used is that the data file is expected to have all the features that are required for that particular run. A very simple one line Awk program can be used to generate new data files containing all the features we wish to use for a run from the original data set as has been shown below:

```
# To create a data file consisting of f1, f2, mf1 and mf2
awk ' { print $4, $5, $8, $9 } ' f0_mf3 > f1_f2_mf1_mf2
```

The **Hillenbrand** data set has three features F1, F2, and F3. Seven vowels have been selected, and a new data file has been created. The data, as before, has been normalized. The only difference between the Peterson and Barney data set and the Hillenbrand data set is that, in the case of the latter, the formant values correspond to the transitions of the formants during the utterance of one vowel, whereas in the case of the former, the formant values represent frequencies in the steady region after the utterance of the /h/ and before the utterance of the /d/. The Peterson and Barney data has been used for training and testing while the Hillenbrand data has been used only for testing using the weights obtained from training the network with the Peterson and Barney data set.

4.2 User Manual

4.2.1 Xview Header Files and Libraries

All the Xview (version 2) files are located in the directory **/home/stu2/g3/jdu4855/X_stuff/lib/xview2**. The header files for the Xview2 programs can be found in the sub-directory "**build/include**" under the above directory. The following libraries are required for the programs

* **xview** library (Standard Xview Library)
* **olgx** library (Open look Graphics Library)
* **X11** library (Xlib Library)
* **math** library (Standard Math Library)

4.2.2 Source Code

The source code for all the programs, the data files and the makefiles may be found on an attached diskette.

4.2.3 The Environment

The neural net can be run under **SunOS, System V** or **Ultrix** operating systems. The net can be run with the graphical user interface only on a machine that supports Xview (Release 2) and Openwindows (Version 2) and operates under SunOS 4.1.1.

4.2.4 Compiling and running the programs

In order to run the programs you will have to do the following:

* "make" the correct executable for the neural net, if it is being run without the interface. The targets may be the following:
    * **make u3b2** for running it on a 3b2
    * **make uSun** for running it on a Sun workstation
    * **make uUlt** for running it on an Ultrix based machine

* The target that would be created after "make" has completed its processing would be called **kohonen**. The neural net can then be run by typing in **kohonen** at the shell prompt.

* If the machine you will be running this program on supports Openwindows, then you will have to use the **imake** utility to generate the makefile. The following command is what you would use for **imake** (Parts of this command would not be necessary when all the Xview files are available in their normal directories).

  Use "**imake -DUseInstalled**
  -I /home/stu2/g3/jdu4855/X_stuff/lib/xview2/config"

  The above directory contains the templates that imake requires for generating the makefile.This command will create the makefile for the neural net and the graphics program that is used to display the phonotopic maps. After the makefile has been generated, typing make will create the two executables **kohonen** and **kohonen_map.** In order to run the kohonen_map program you will need to have the data file that has all the information about the node activations in the current directory.

4.2.5 Running the Neural Net

The neural net as explained above can be run with or without the graphical interface. To start the neural net, type in **kohonen** at the shell prompt. If the neural net is being used **with** the interface, the first frame that is brought up has the following major options:

* Start training: to start training a new training session
* Continue training: to use existing weights from a previous run and continue training
* Test the trained net: use existing weights from a trained net
* Display Maps: draw and display the different phonotopic maps
* Quit: to quit the application.

A screen dump of the frame as it appears on the screen is shown below (Figure 18)

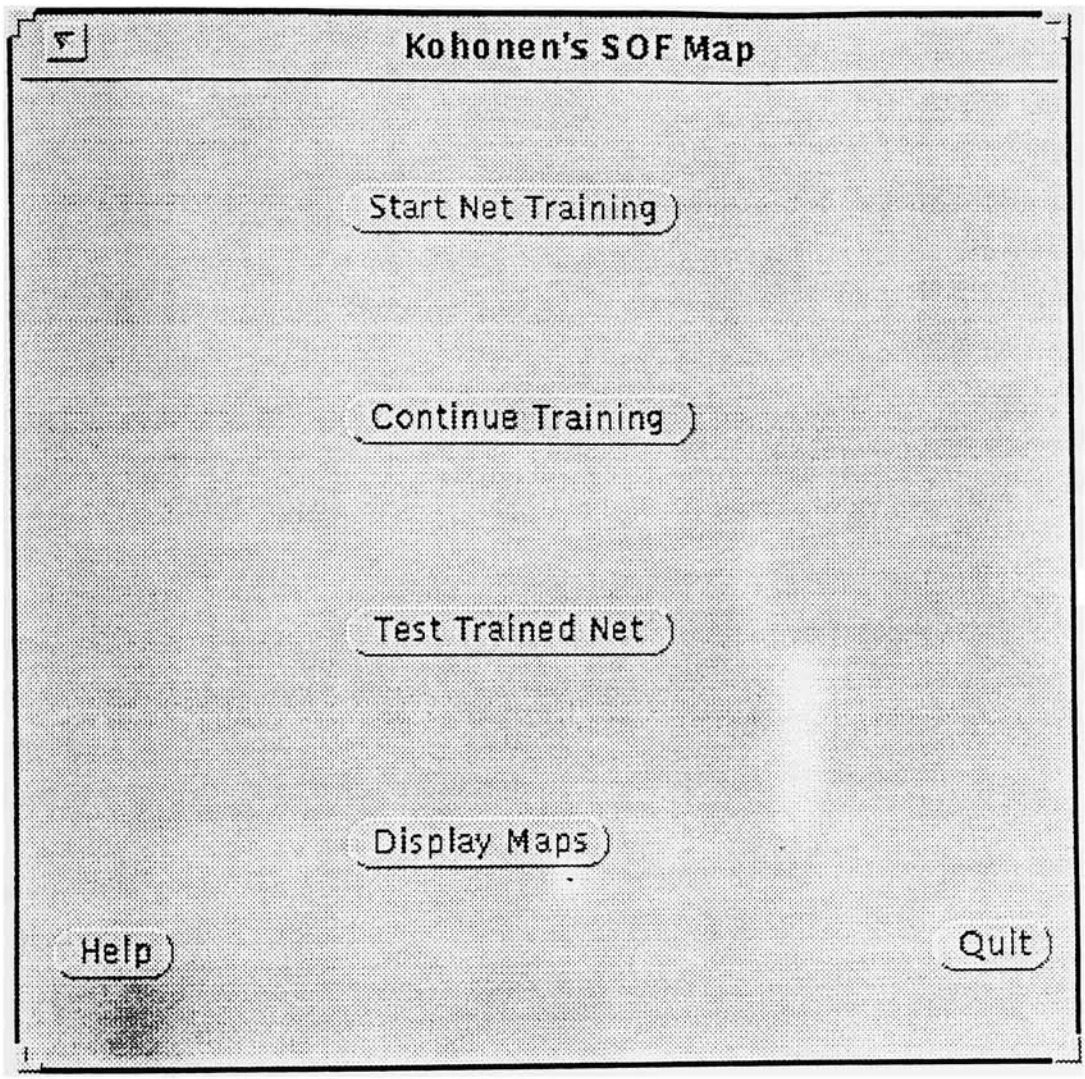


Figure 18: Screen dump of Frame "Kohonen's SOF Map"

The user initiates the start of the training process by clicking on the "Start Training" button. A notice will be brought up that asks the user to provide parameters, and when done, to click on the pushpin to bring the frame down and start training. The frame that is presented then has the above parameters in the form of "Test Items". The user has the option of clicking on the buttons for changing the values. If the upper or lower limit is reached for a parameter, a notice is displayed to that effect. The user is also allowed to type in the values. If what was typed in was unacceptable to the system (numbers greater than the maximum or lesser than the minimum), the parameter is reset to its default value and a notice is put up. The connection weights initially can be set to random numbers or can be set to the same value throughout the net. This option is present to the user as a "choice item". The files that are to be opened for reading are immediately opened. If the operation fails, a notice is displayed to that effect and the user is expected to enter another name. Once the user is done entering all parameters, clicking on the pushpin causes the frame to be unpinned and the net starts to run. If the net has to be interrupted, CTRL-C may be used. Clicking on the quit button causes the application to be terminated after seeking confirmation from the user. The user should be forewarned that it takes a little while for the application to respond to the mouse and button events and focus attention to the user's requests. In the absence of any input from the user, the default values that are supplied are used and these are the values that are displayed initially when the frame is brought up.

The following are the parameters that are requested from the user. A screen dump of this frame has been shown in the following page (Figure 19).

* The number of input features. This specifies the number of input nodes/features that are necessary for the network.
* The number of nodes in the X-dimension of the output layer.
* The number of nodes in the Y-dimension of the output layer. These two parameters together specify the topology of the output layer of network.
* The initial neighborhood. This specifies the radius (number of nodes) around the winning node that participate in the weight update.
* The final neighborhood. This specifies the final value to which the radius is decremented when the training stops.

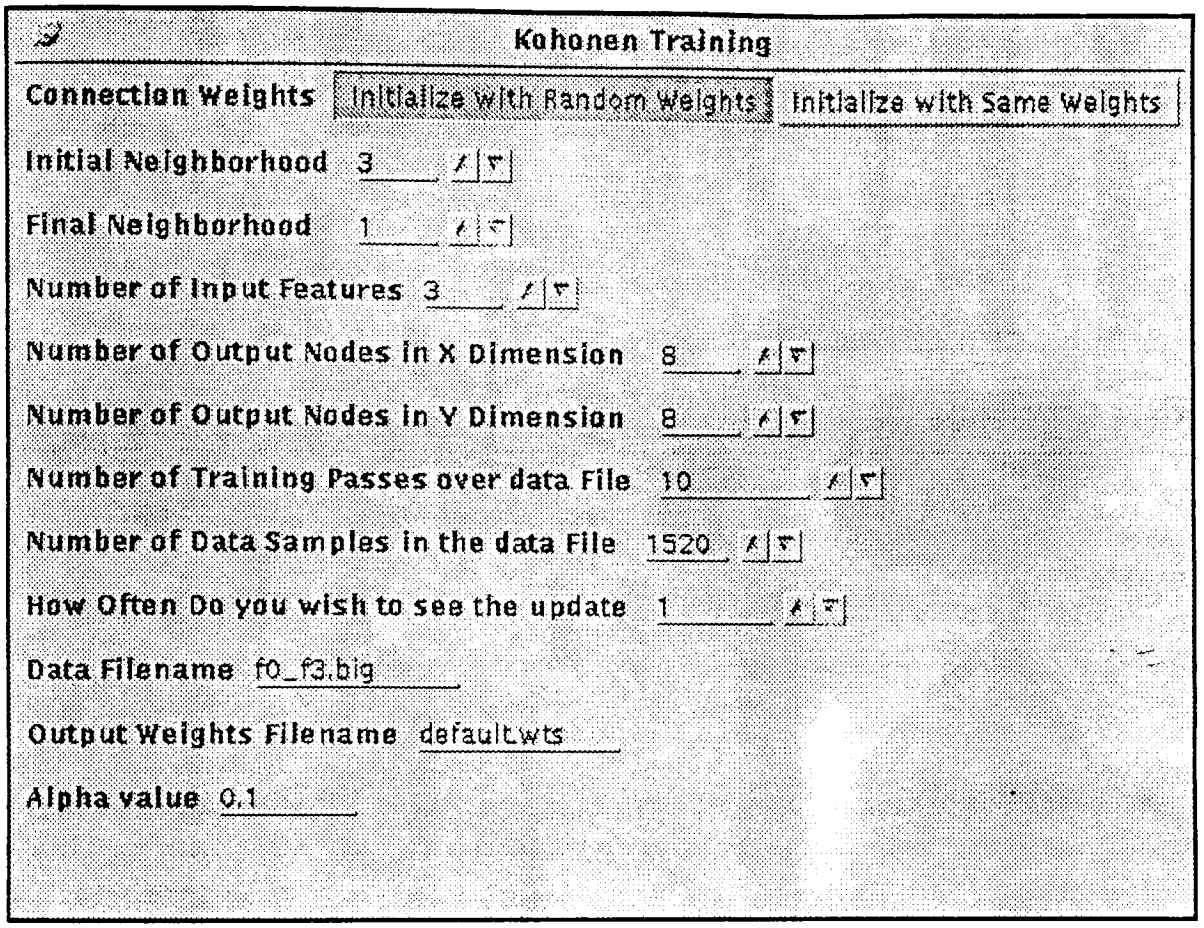


Figure 19: Screen dump of frame "Kohonen Training"

* Data filename. This file has all the input features stores in it. It is assumed that the data file has all and only the required input features in it.
* The initial α value. This is the gain term that decreases towards 0 with time from this initial value. Typically, α should be between 0.0 and 1.0

* Number of training passes. This is the number of times the user wishes to train the net with the data file. Each pass constitutes an **epoch.**
* How often the weights are to be written to file. This specifies how often the weights are to be written to be file along with all current net parameters, how often the testing is to be done and the tables and a confusion matrix printed, and how often the node activation information is written to file so that it could be used for drawing the maps using the Kohonen_map program.

If the user is continuing the training process, the following information is requested from the user:

* Input weights filename
* Data filename
* Output weights filename

A window dump of this frame has been shown in the following page (Figure 20).

If the interface is being used, the user continues training by clicking on the "Continue Training" button. The user is again informed with a notice about the next frame. Once all values are supplied, the files are opened for reading. If this is not possible, a notice to that effect is displayed. This frame has a "Confirm Parameters" button which when pressed causes the read values to be displayed. The user then has the option of modifying the values. Once these have been provided, the program attempts to open the weight file and read in all the parameters. If the user then clicks on this button, the values read from the file are presented. The following may be changed before the run commences.

* Initial Neighborhood
* Final Neighborhood
* Number of training Passes
* Current pass Number
* Number of data samples in the data file
* Current value of α

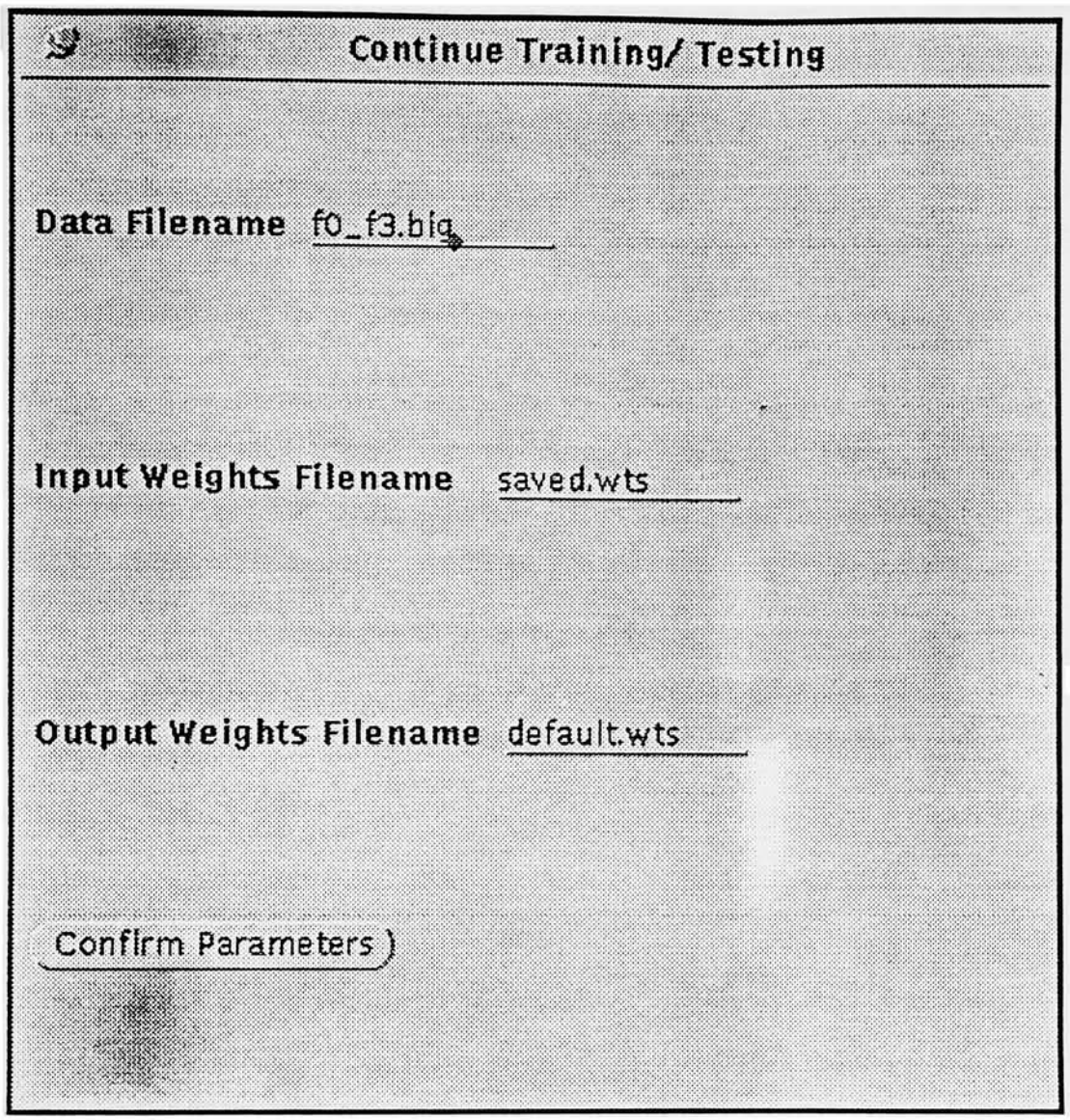


Figure 20: Screen dump of frame "Continue Training/ Testing"

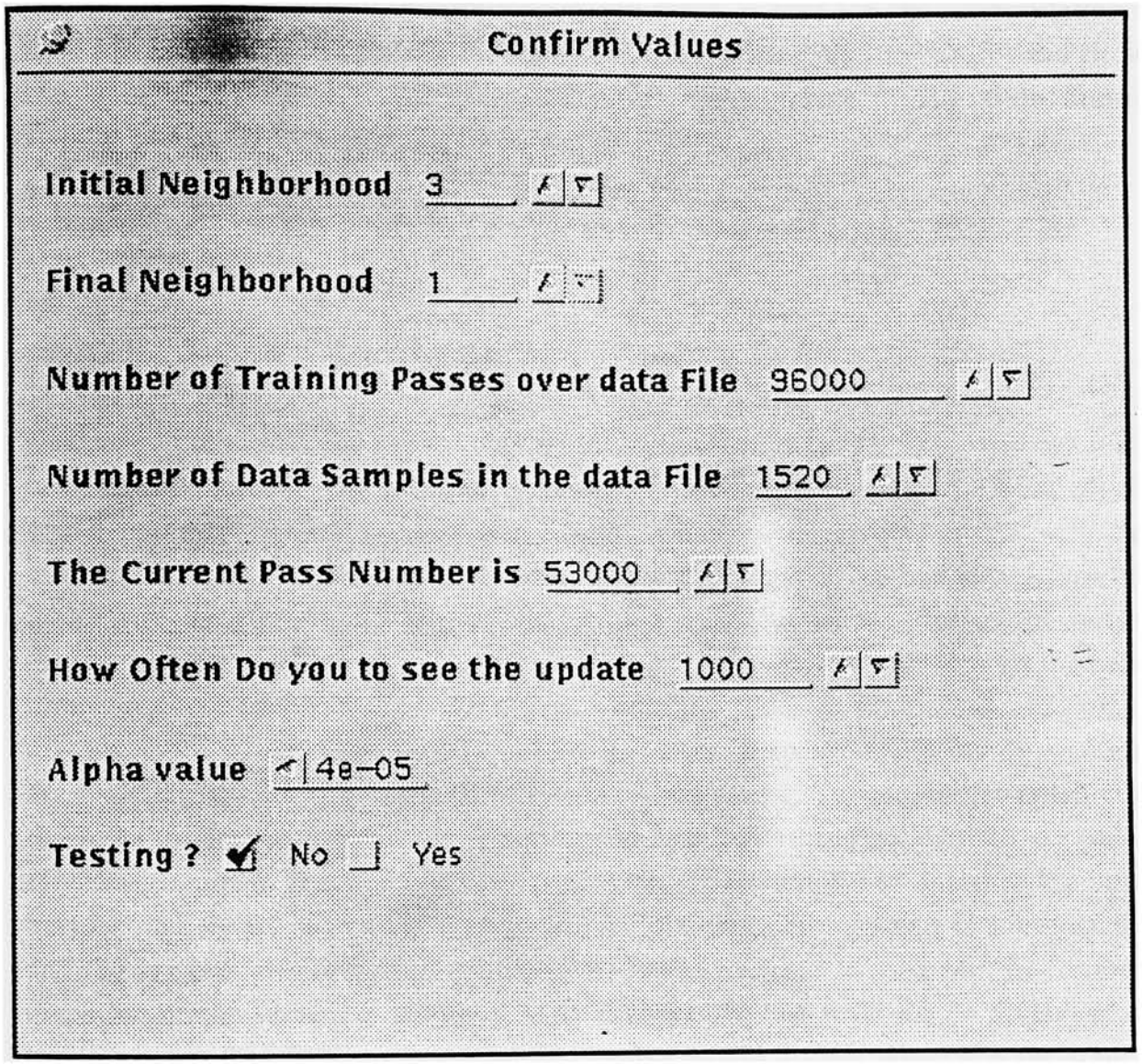


Figure 21: Window dump of frame "Confirm Values"

The parameters read from the data file that cannot be changed are the ones that specify the net topology, the decay rate for the neighborhood, δ, and the current neighborhood. The values that are displayed and changeable are the ones described above. Clicking on the pushpin causes this frame to go away. Clicking on the pushpin of "Continue Trainer/Tester" frame causes the frame to be unpinned and the training to continue.

In order to test the neural net with a set of weights obtained after the training process, the procedure to be followed is the same as that for the continuation of training, except that for the continuation of training, the "No" option for the choice box labelled "Testing ?" in the "Confirm Input Frame" frame is clicked on, and for Testing the "yes" button in the choice box is selected.

A screen dump of this frame has been shown in the previous page (Figure 21).

4.2.6 Using the vowel map interface

This section explains the capabilities of the **kohonen_map** program and what the user can do with it. The program takes for input the node activations for all the vowel samples present in the data file. The neural net creates a file with this information along with information about the topology of the net every time the tables and the confusion matrices are printed out. The dimensions of the output grid during most of the tests were 8x8. This program creates a grid with the same dimensions as the one used during the training process. When the program is invoked from the shell prompt by typing **kohonen_map &**, a number of messages are printed indicating the different stages of processing. These are the creation of the color map segments, the creation of the base frame, the creation of the panel, the creation of the canvas on which the maps are drawn, the creation of the panel buttons and choice items, the creation of the pixmaps, the opening of the data file, and the creation and setting of the icon. The first set of data is read and the different maps are drawn on to the pixmaps. Once this is done, the map can be displayed on to the screen. Figures 22 shows how the display would appear when this program is running. The entire window may be considered to be made up of the frame that encompasses the panels and the canvas. The panel is the top section of the window that has all the buttons on it. The canvas is the part below the panel that has the maps drawn on to it. There are two scrollbars attached to the two sides of the frame that can be used for scrolling through the parts of the canvas if all of it was not visible at once. The top row of buttons in the panels labelled **Vowels** correspond to the ten vowels and an eleventh for **"no activation"**. The row of choices immediately below labelled **Colors** indicate the colors associated with these vowels. The two rows of buttons are both exclusive, meaning **only one** choice can be selected at any one time. The names of the choice appear on the respective buttons.

There are also two buttons for the foreground and background along with two colored boxes for the colors to which they are currently set. The user does not have the option of changing the colors associated with the vowels dynamically in the present version of the interface. The third row of buttons labelled **View** are the choices for the various maps that can be displayed on the screen. Each represents a different interpretation of the same data. There are two more buttons that are labelled **next pass** and **quit**. If any of the buttons on the topmost row is clicked, it brings up a **notice**, which displays the name of the vowel, the vowel number it corresponds to in the data file, an example of the vowel in a word, its arpabet symbol, and the color that represents it. Clicking on the **ok** button makes the notice disappear. This is true also for the choices associated with the foreground and background. Clicking on the buttons in the second row, that represent the different colors for the vowels, causes another notice to be brought up. It indicates the name of the color, the vowel it is attached to, and informs the user that the screen will be redrawn to show all occurrences of that vowel for that pass in the output grid. The amount of color present in each box indicates the percentage of the total activation in that node for that vowel. In all nodes where this color did not appear, the background color is used for painting the node. If the same button were to be clicked once more, it is unselected and the map that was on display before this button was depressed is redisplayed.

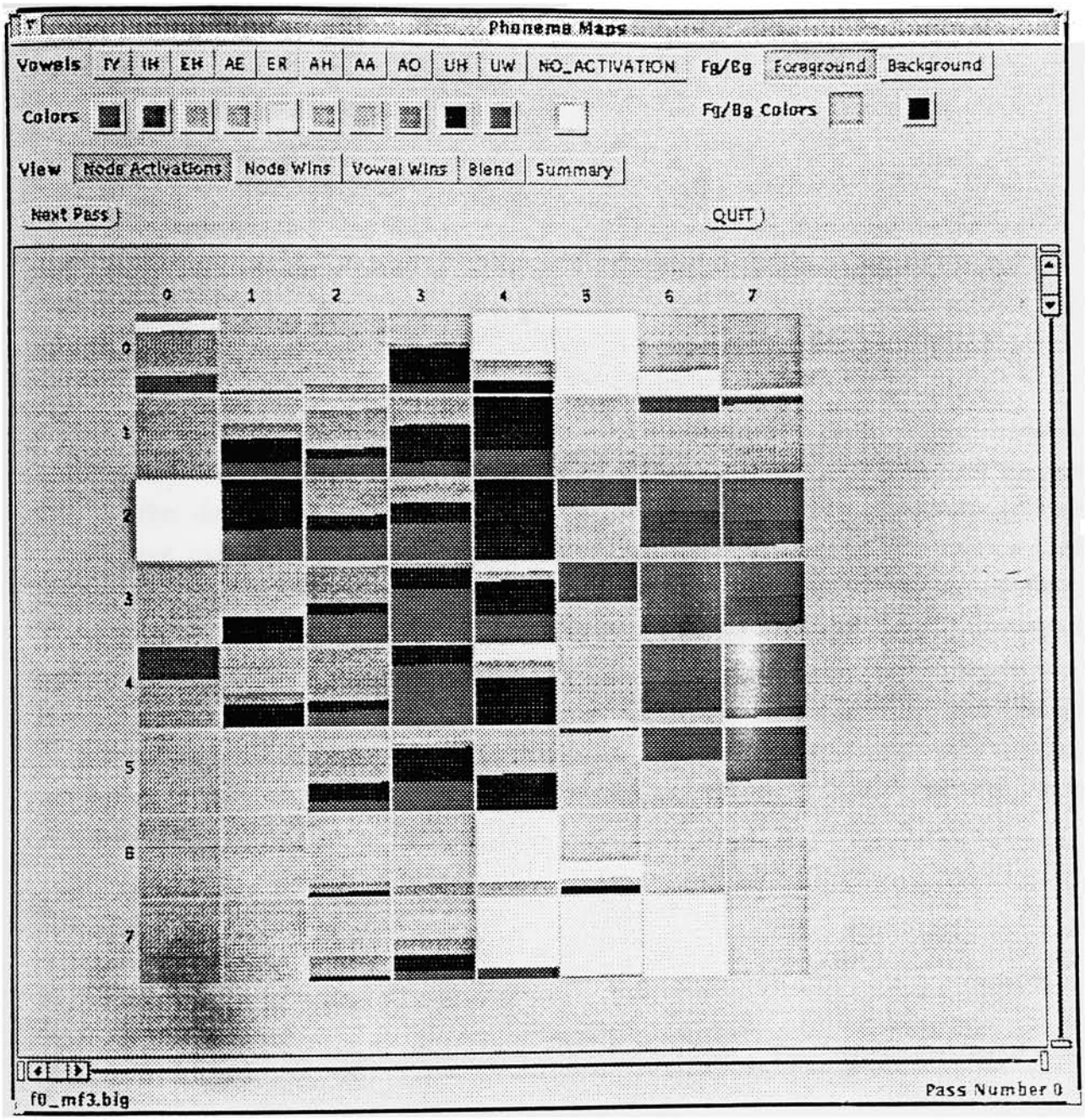


Figure 22: Screen dump of the Map Drawing Program

4.2.6.1 Vowel Maps

There are five choices under **View**:

* **Node Activations**
* **Node Wins**
* **Vowel Wins**
* **Blend**
* **Summary**

4.2.6.1.1 Node Activations

The choice that is called Node Activations is one of the five maps that is created. The pixmap associated with this choice is the first that is created for every set of data and is the default map that is displayed initially. This map displays all the activations associated with all the nodes. If a node has more than one vowel activating it during that pass, then that node has more than one color present in it. The amount of color present represents the **percentage** of activations each vowel had of the total number of activations for that node. When the choice is selected, (by clicking on the button), the pixmap is copied onto the screen. The use of pixmaps makes the redisplaying of the maps very quick. If a particular node was not activated during that pass, it is painted white, which is the color for "**No Activation**".

The algorithm is as follows:

```
for each node in the output grid
{
        determine the total number of activations  for that node;
        for each vowel that activated the node
        {
                compute the percentage the vowel represents out of the total;
                compute the number of pixels it occupies out of the 3600 for the
                entire grid based on the percentage;
                set the foreground color and paint that many pixels;
        }
```

}

The size of each node is 60x60 pixels in all the maps except the summary map, which uses a 30x30 pixel square.

4.2.6.1.2 Node Wins

This choice creates a pixmap by determining which vowel each node in the output grid became most sensitive to, and labelling it accordingly. Every node's activation due to all vowels is examined, the vowel that has had the maximum number of activations is considered the winner, and the node is painted in the color of the winning vowel. In case of a tie for the maximum, the node is divided into the appropriate number of segments, and each segment is painted with the colors of the winning vowels.

The algorithm for this map is as follows:

```
for each node in the output grid
    {
      determine the total number of activations  for the node;
      determine the number of vowel(s) that caused maximum response
      for that node;
      divide the node into as many segments as there are winning vowels;
      for each winning vowel
        {
            set the foreground color and paint its segment with the
            vowel's color ;
        }
    }
```

4.2.6.1.3 Vowel Wins

This map is created by determining which processing unit (output node) each vowel sample cause maximum response. This map is created by picking out the node that has responded the maximum number of times for each vowel and painting that node with the color associated with that vowel.

The psuedo-code for generating this map is as follows:

```
for each vowel in the data set
  {
      determine the node at which this vowel caused the maximum
      response;
  }
determine whether this node represents more than one vowel;

for each of the vowels claiming a node
{
  set the foreground color appropriately and paint the node  in
  the proportion of its activation;
}

set the foreground color for the other nodes to the background color,
and paint the node;
```

4.2.6.1.4 Blend map

This is a pixmap that is created to represent the strength of each vowel over the entire map. If a particular vowel has caused a very strong response in an output node and there are no other contenders for that node, the node retains the color of that vowel. If more than one vowel has caused a response in an output node, the color that is used to paint the window is generated as a weighted average or blend of the vowels' colors.

For example, if **Red**, **Blue** and **Green** are the red, blue and green components of the pixel that is generated dynamically as a combination of the colors that have activated a node, the **Red**, **Blue** and **Green** values are computed as follows:

```
initialize red=blue=green=zero
for each output node in the grid
{
  for each vowel that has caused a response to that node
   {
        Red  +=
              percentage that this vowel corresponds to out of all activations
              for this node x  its color's Red component;
        Blue  +=
              percentage that this vowel corresponds to out of all activations
              for this node x its color's Blue component;
        Green  +=
              percentage that this vowel corresponds to out of all activations
              for this node  x   its color's Green component;
   }
  Request for allocation of the pixel represented by  {Red, Blue, Green} from the
  X server;
  set the foreground and paint the grid;
}
```

The blending is used to present a picture about the dominance of each vowel over the entire map. It is interesting to note how this changes as the maps evolves with training. If a particular vowel does not have a strong influence on a node and a number of vowels activate it, the blend produces a totally new color. If one of them has had a very strong influence, the node gets painted in a lighter shade of that vowel's color.

4.2.6.1.5 Summary Map

This map is a summary combining the four maps explained above. It contains each of these in half-sizes (node size of 30x30 pixels). It gives the user the opportunity to examine all the maps at once. The only difference from the maps explained above

and the ones drawn here is that the **Node Wins** map here also gives the number of activations associated with each node. This gives the user the ability to judge how much, "in quantitative measure", the colors in each of the nodes in the Node Activation map represents.

4.2.6.1.6 Next Pass

If the leftmost mouse button is clicked on the **next pass** button, the program reads the next set of data from the data file, recomputes and redraws the pixmaps, and displays the Node Activations map (this is the default) when done. The computation and redrawing of the pixmap takes about a minute; while this is being done, the **next pass** button is grayed, indicating that the application is processing information. Every time the pixmaps are redrawn, the left footer for the frame is updated to indicate the current pass number. If the end of the data file is reached, the file is rewound, and the maps are redrawn starting with the first set of data in the data file.

4.2.6.1.7 Quit

This button, when depressed, causes the application to terminate. A notice is displayed, and if the user confirms wanting to quit, the application exits gracefully.

Figure 23 shows a screen dump of the summary map.

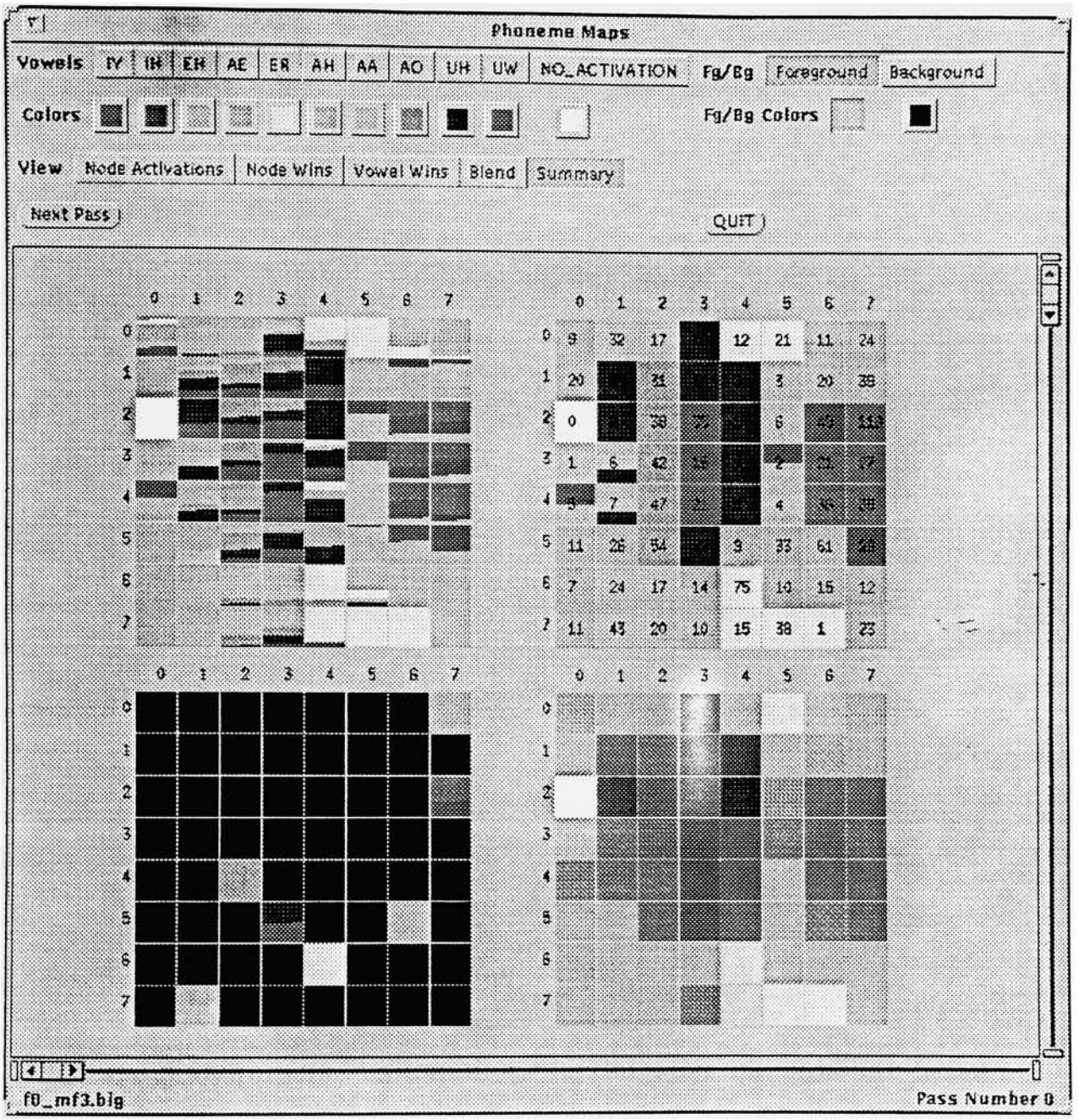


Figure 23: Window dump of the Summary Map

4.3 Architecture

This section describes my algorithm for producing Kohonen's self-organizing feature maps along with explanations of the different programs that have been used.

4.3.1 Algorithm for Producing Feature Maps

The following are the steps which produce self-organizing Feature maps. Use Figure 24 for reference for the terms used below:

Step 1: *Get input from the user; compute the value of the decay and the value of* δ.

Step 2: *Initialize weights.*

Initialize weights for the connections from all inputs to all output nodes to small random values. Set the initial radius of the neighborhood as shown in Figure 24.

Step 3: *Present input vector.*

Step 4: *Compute the distance to all nodes* (i.e., the response strength of each node).

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

$$d_j = (x_i(t) - w_{ij}(t))^2$$

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

Step 5: *Select output node with minimum distance* (i.e., the strongest response).

Select as node j^* the best node with minimum d_j.

Step 6: *Update weights to node j^* and its neighbors.*

Weights are updated for node j^* and all the nodes in its neighborhood defined by $NE_j^*(t)$ (Figure 24).
The distance from each node in the neighborhood to the winning node is computed. Let this be **d1.**
This distance is then used to compute **ratio** = d1 / neighborhood.
If the distance is greater than the current neighborhood radius, ratio is saturated to 1.
New weights are computed by using

$$w_{ij}(t+1) = w_{ij}(t) + (1- \text{ratio}) \times \alpha \times (x_i(t) - w_{ij}(t))$$

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

α is a gain term ($0 < \alpha < 1$) that decreases with time
Decrement the neighborhood by decay.
Decrement the gain term α by δ.

Step 7: *Repeat by going to step 2*

Weights will eventually converge and will remain constant after the gain term in step 6 is reduced to zero. At this stage the algorithm terminates.

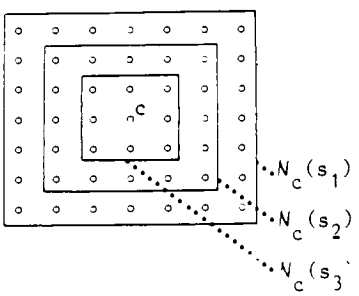


Figure 24: Neighborhood (*adapted from [Lippmann88]*)

4.3.2 Computation of Number of Training Passes

The number of training passes is computed in the following manner:

total_passes = passes x Output_X x Output_Y

where

passes is the total number of passes
Output_X is the dimension of output layer along the X-axis
Output_Y is the dimension of output layer along the Y-axis

training_samples = total_passes x data_samples;

where

data_samples is the number of data samples in data file.

4.3.3 Computation of δ

$\delta < 1$ is the factor by which α is multiplied to reduce it towards zero. Since the final value of α has to be a very small number (close to zero), the values that I have used for final the value of α have been 0.0000001 and 0.005.

$$(\delta)^{\text{training\_samples}} = \frac{\alpha_f}{\alpha_i} \quad ;$$

where

α_f is the final value of α
α_i is the initial value of α

Therefore δ would be computed by the formula:

$$\delta = \exp\left\{ \frac{\log\left(\frac{\alpha_f}{\alpha_i}\right)}{\text{total number of training samples}} \right\}$$

The value of α is decremented after processing for every input vector by the formula:

$$\alpha = \alpha \times \delta ;$$

This ensures that α drops to its final value once the net has been trained.

4.3.4 Computation of Decay

Decay is the factor by which the neighborhood is decremented every time from its initial value.

$$(\text{Decay})^{\text{training samples}} = \left\{ \frac{(\text{neighborhood})_{\text{final}}}{(\text{neighborhood})_{\text{initial}}} \right\};$$

where

$(\text{neighborhood})_{\text{final}}$ is the final value of the neighborhood (radius)
$(\text{neighborhood})_{\text{initial}}$ is the initial value of the neighborhood(radius)

Thus:

$$\text{decay} = \exp\left\{ \frac{\log\left(\frac{(\text{neighborhood})_{\text{final}}}{(\text{neighborhood})_{\text{initial}}} \right)}{\text{total number of training samples}} \right\}$$

The value of the neighborhood is decremented after processing for every input vector by the formula:

$$\text{neighborhood} = \text{neighborhood} \times \text{decay} ;$$

This ensures that the neighborhood drops to its final value once the net has been trained.

The following diagram (Figure 25) gives the flow of control in the neural net as it appears in the code.

```
compute the total number of iterations  and training_samples;

compute delta;

compute decay;

for( pass_num = One to total_number_of _passes )

   {
     for(input = One to number of data_samples in file )

          {
                read input vector;
                calculate distance;
                adjust weights;

                alpha = alpha * delta;
                neighborhood = neighborhood * decay;

          }

       if( ( pass_num  % how_often_wts_are_to_be _written ) == 0)

           {
                 determine all node activations for one pass of the data file;
                 label all nodes;
                 create confusion matrix by testing the data file;
                 print tables indicating classification for the test pass;
                 print confusion matrix;
                 write weights to file;
           }


    }

 write weights to  file;
```

Figure 25: Flow of Control For the Neural Net

4.3.5 Programs

main.c

This is the main program that is used when the net is used without the graphical interface. It gets input from the user, validates it, dynamically allocates memory for the arrays (by calling the **mem_alloc** function), initializes the weights (by calling the **initialize** function) or reads weights from a file (by calling the **read_initial_wts function**), opens all files and passes control to the function **kohonen**, from where the actual functioning of the net starts (Figure 25).

graph_kohonen.c

This is the main program that is used with the graphical user interface to run the neural net. It has all the Xview based code to create the frames, panels, buttons and the call back routines these objects need. This program also has the code for controlling the action of the neural net (the **kohonen** function referred to above). Once the interaction with the user for the input is over, the net starts to run in the usual manner.

init.c

This has the function that initializes all connection weights to random values. It is called when a new training session is started. The function **drand48()** is used as the random number generator.

memory.c

This has the program that allocates memory for the arrays that are used in the programs. The only parameters that this function depends on is the net topology, i.e. the number of input features and the size of the output grid.

control.c

This has the function that controls the running of the net and is best described by the diagram presented earlier (Figure 25).

data.c

This has the function that reads the input vector from the data file. In one of the variations of the neural net, the use of this function has been obviated by reading in earlier all the data into an array and then using the array to obtain the input vectors. This substantially reduced the amount of disk I/O that was being done (especially when running the net from a client).

calc.c

This has the function that calculates the distances between the input vector and the weight vector for each of the nodes. It returns the node that has the minimum distance for the given input. The method by which this calculation is done has been explained in the algorithm presented earlier.

adjust.c

This is the function that is called to adjust the weights once the node that has the least distance with respect to the input has been determined. The method by which this is done has been explained in the algorithm presented earlier.

write.c

This has the function that writes all the information pertaining to the current state of the net to a file. The name of the file is of the form **current_pass_number.wts**. The information that is written to it consists of the network dimensions, the weights for the connections, the α and δ values, the neighborhood and decay values, the current pass number , the total number of passes, the initial and final neighborhoods. This is to allow continuation of training at a later time.

read.c

This has the function that reads in information from the file when training is continued from an earlier run. After reading the data the user is given the option of reassigning values to some of the parameters that were read in for use in that training session.

my_tab.c

This has the function that prints the tables summarizing the node activations associated with the input. These tables are printed out every time the weights are to be written to file. This function creates a file whose name is of the form **current_pass_number.tab** and writes to it the node activation information for that pass along with the dimensions of the net. A number of these files are later combined and used by the graphics program to draw the vowel maps.

confuse.c

This is the function that prints the confusion matrix. The rows denote the actual classes. The columns denote the classes as the net determined it to be. An error in percentage is also printed out that gives an estimate of the error in the classification. This is printed out after the tables explained in the previous section have been printed out.

kohonen_map.c

This contains the code for drawing the maps using the data produced during the training. The methods by which the maps are drawn have been explained in the earlier sections.

4.4 Design Decisions and Trade-offs

A present limitation on the use of the neural net with its graphical user interface is the availability of a machine that supports Xview. The use of this net on a machine that does not support Xview or Openwindows forces the use of tabular outputs to display the training and testing processes.

The two parts of the GUI as described above have been kept separate due to limitations imposed by the system. The workstation being used is a Sun 3/60 running under SunOS 4.1.1. The response on the machine when the neural net is running is very poor, due to the computational overhead of the net. This makes the possibility of examining the different maps for a particular set of data while the net is running very remote. The availability of a faster machine should considerably improve performance and the response time, and make the application's response to user-initiated events much better. The integration of the two modules when faster systems are available would be useful and realizable. It would then be possible to run the neural net and the graphical interface on the fast machine and use a color monitor on the network to display the results. The programs have been designed with this future integration in mind.

The separation of the interface into two parts also gives the user, besides improved performance from the system, the ability to use data from a number of passes all at once. This helps in seeing the evolution of the maps and the movement of the vowels on the grid.

In the drawing of the different maps, pixmaps were chosen as the **Drawables**. Although the generation of the maps takes a while (when the pixmaps are being redrawn) switching between different maps once they have been drawn is very fast. This is far more efficient than drawing the maps on the canvas when the button is pressed by the user.

4.5 How is my Algorithm Different?

The algorithm I have used is different from the standard algorithm found in the texts. A number of the changes that were incorporated were at the suggestion of Dr. David Van den Bout of North Carolina State University. The most important changes were the following:

Investigating the effect of **wraparound** and its absence during the update of weights in a neighborhood. The presence of wraparound means that the neighborhood around the winning node would span past the ends of the grids to include the nodes from the other ends. The absence of the wraparound means that those nodes that fall outside of the boundaries of the grid would be ignored in the update. When the wraparound effect is allowed, the presence of phonotopically similar nodes on the borders of the grid can be observed. Wraparound makes the movement of the vowels during the learning stages easier. This helps in making a better classification.

Accounting for the distance of the winning node from its neighbors during the weight update. The nodes that are farther away get less of an update than the ones closer to the winning node. This ensures localization of the activations of the nodes in the net.

The rate at which the radius (neighborhood) starts to decay. The rate at which the neighborhood decreases towards the final value is slow and gradual. Another method that could be used, but would constitute an abrupt decrease, would be to decrement the neighborhood by 1 after training for a certain number of passes.

Rate at which the gain term α is decremented ever time. This is done by the method explained in the earlier section when the algorithm was presented. Two lower bounds for α that were used: 0.0000001 and 0.01 of the initial value of α in the training sessions, respectively.

Computing the total number of passes.

5.0 Simulations

5.1 Experiments Performed

The Experiments that I performed may be divided into four categories.

They are :

* **Random Numbers**
  *Classification of two data points using the net
  *Classification of three data points using the net

* **Two vowels**
  * 2 like vowels from the Peterson and Barney Data set
  * 2 unlike vowels from the Peterson and Barney Data set

* **All ten vowels from the Peterson and Barney data set.**

When the training and testing was done with the ten vowels, the following were variations were tried to determine their effect on the training process.

* Varied the initial value of α
* Varied the rate of change of α
* Wraparound versus no wraparound
* Different feature sets
* Grid size
* Radius (neighborhood).

* **Testing using the Hillenbrand data** consisting of formant values for seven vowels taken at 10 msec intervals.

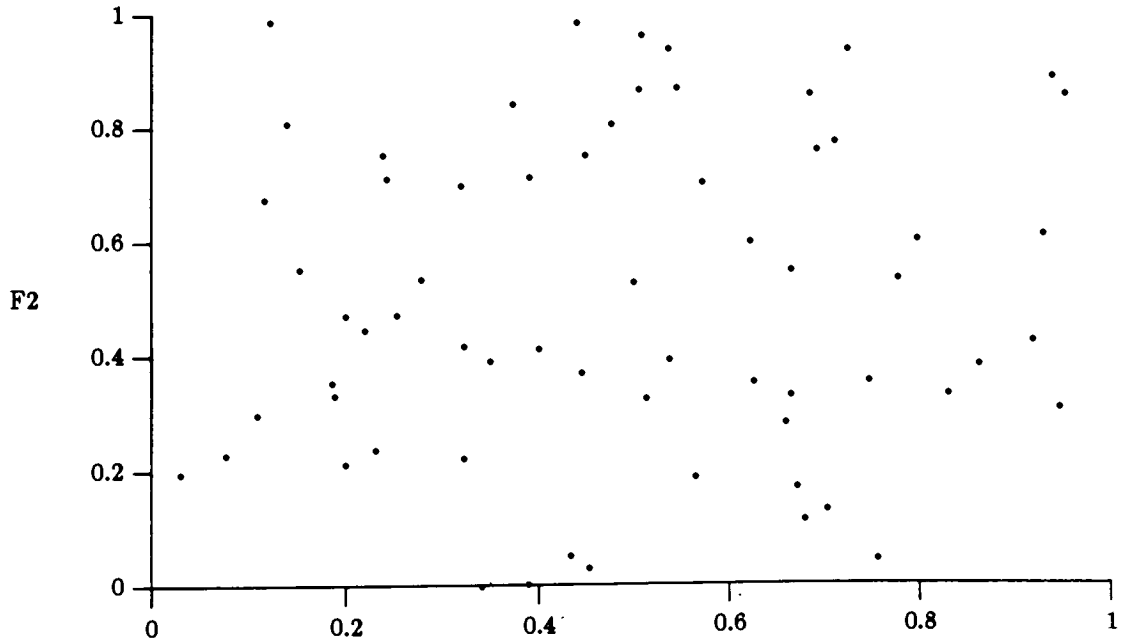
5.2 Simulation Results

The Kohonen neural net was tested with artificial data created using random numbers. This was done to study the working of the self-organizing process. Two tests were conducted. The first used points generated around (0.2, 0.2) and (0.8, 0.8). Two distinct clusters were observed. The second test used three sets of points generated around (0.0, 0.0), (0.2, 0.2), and (0.8, 0.8), and, as expected, three distinct clusters were observed. The Kohonen net seemed to perform well with the numbers generated randomly.

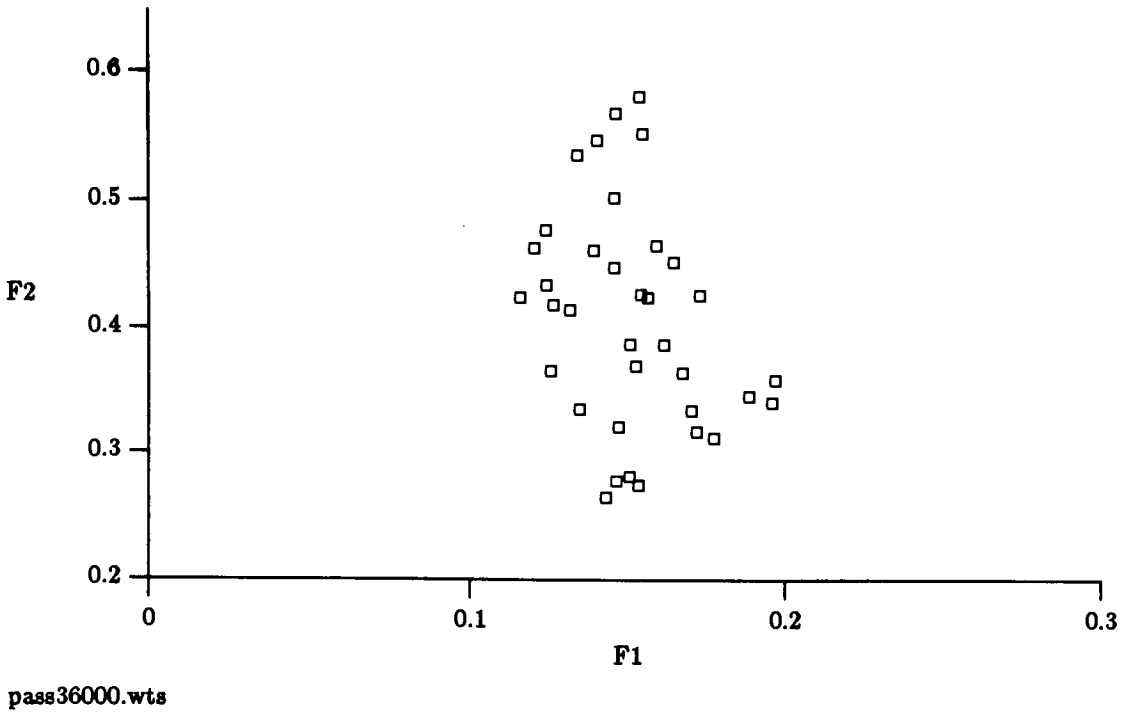
The net was trained and tested using two vowels from the P&B data set. The two vowels were EH and AH, which are relatively distant phonotopically. The three graphs show:

(a) distribution of weights initially ;
(d) the distribution of the weights after 36000 passes;
(c) the distribution of the data for the two vowels.

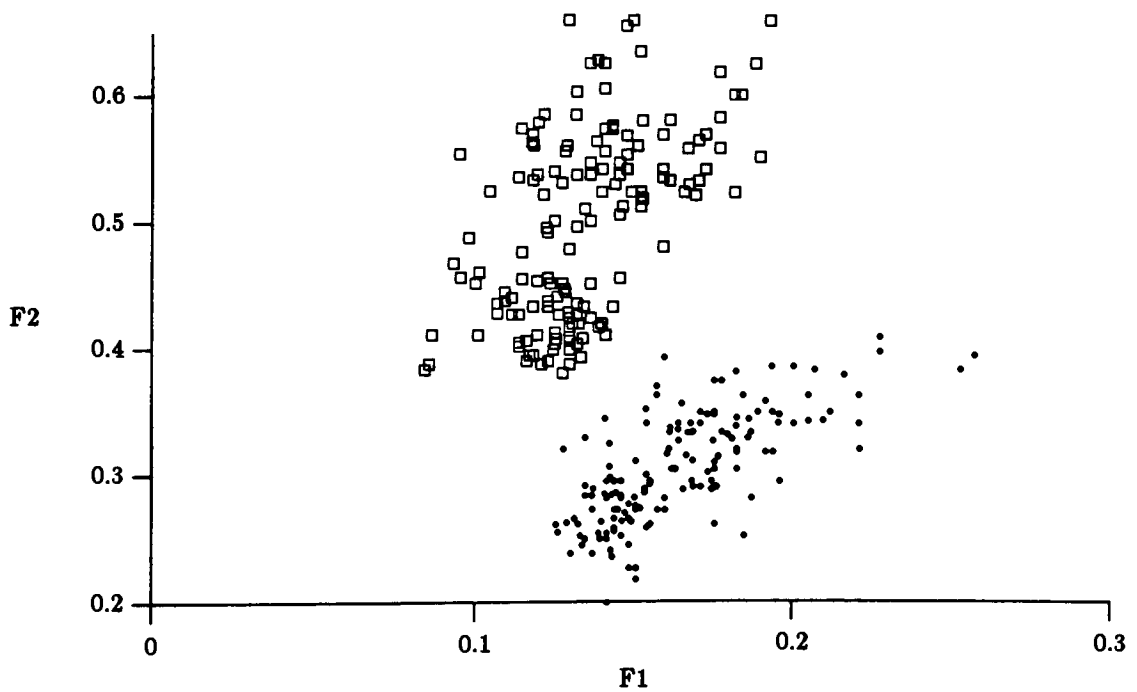
It can clearly be seen that the weights tend to distribute themselves so that entire input space is sampled and represented. The same test was performed for two vowels AH and AA, which are close to one another from a phonemic standpoint. The classification by the net showed a fair amount of confusion. These tests indicate that the net performs well and that the errors that are being observed are logical.



Graph 1: Initial weights distribution



Graph 2: Weights after 36000 passes for two vowels AH and AA



Graph 2: Distribution of data for AH (box) and AA (dot)

The results for the the tests with all the **10 vowels** from the P&B data set have been tabulated below.

Features	Wrap Around	Pass Number	Percentage Error	Data File
F0-F2	YES	0	80	f0_f3
		1	35.7	
		49600	20.1	
	NO	1	29.3	
		49600	30.1	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F0-MF3	YES	0	71.6	f0_mf3
		1	37.4	
		49600	33.2	
	NO	1	38.9	
		49600	35.8	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F0-MF2	YES	0	67.5	f0_mf3
		1	36.5	
		49600	34.0	
	NO	1	36.5	
		49600	335.8	

α = 0.1 to 0.0000001

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F2	YES	1	33.8	f0_f3
		30000	27.7	
		55000	27.6	
	NO	0	55.5	
		1	33.6	
		30000	29.4	
		60000	29	

α = 0.9 to 0.0000001

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F2	YES	0	55.1	f0_f3
		1	32	
		24000	29.8	
		49600	27.3	

α = 0.5 to 0.0000001

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F2	YES	0	55.1	f0_f3
		1	31.8	
		20000	28.5	
		49600	24.7	

α = 0.5 to 0.005

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F2	YES	0	69.6	f0_f3
		1	31.8	
		20000	31.1	
		49600	24.7	

Neighborhood varied from 4 to 1, 16x16 grid

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F2	YES	0	78.9	f0_f3
		1	22.5	
		20000	20.9	
		49600	19.8	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F0-F3	YES	0	80.7	f0_f3
		1	35.7	
		24000	30.1	
		49600	27.3	
	NO	0	49.9	
		1	38.1	
		24000	34.9	
		49600	37	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-MF3	YES	0	80.2	f0_mf3
		1	42.3	
		24000	36.8	
		49600	34.0	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F3	YES	0	80.2	f0_f3
		1	42.3	
		24000	36.8	
		49600	34.0	
	No	0	71.4	
		1	41.4	
		24000	35.3	
		49600	36.0	

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1,F2,MF1,MF2	YES	0	79.1	f0_mf3
		1	34.3	
		12000	28.1	
		24000	19.7	

α = 0.5 to 0.005, Neighborhood varied from 4 to 1 , 16x16 grid

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1F2mF1mF2	YES	0	76%	f0_mf3
		1	17.9	
		3000	17.2	

α = 0.5 to 0.005, Neighborhood varied from 4 to 1, 16x16 grid

Features	Wrap Around	Pass Number	Percentage Error	Data File
F1-F3	YES	0	80.1	f0_f3
		1	23.2	
		6000	26.0	
		12000	23.8	

The above tables reflect the way the Kohonen net performs under different conditions. Wraparound improves the classification accuracy. The initial value of α also plays an important role. For best results, it is recommended that the value of α be close to 0.5 initially and that the decrease in α be gradual.

A sample confusion matrix is printed for the feature set F1-F3. The net was trained on the P&B data set and tested on seven vowels from the Hillenbrand data set. The confusion matrix shows the accuracy to be 20.6%. The accuracy of the net with all the ten vowels was about 33.1% after 49600 passes (with the P&B data set).

Confusion Matrix for pass number 49600 for F1-F3

Total number of input samples = 1520

Number of Errors in this pass 503

Percentage error 33.1

Rows specify the actual classes, Columns specify the class as identified by the ANN

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	119	33	0	0	0	0	0	0	0	0
1	26	112	13	0	1	0	0	0	0	0
2	7	27	101	6	11	0	0	0	0	0
3	1	2	50	84	9	6	0	0	0	0
4	0	7	6	2	135	1	0	0	1	0
5	0	0	0	0	2	84	54	3	7	2
6	0	0	0	1	0	27	111	11	2	0
7	0	0	0	0	0	0	29	93	5	25
8	0	0	2	0	1	13	4	19	79	34
9	0	0	1	0	0	5	0	17	30	99

Confusion Matrix for vowels in Hillenbrand Data for F1-F3

Total number of input samples = 519

Number of Errors in this pass 107

Percentage error 20.6

Rows specify the actual classes, Columns specify the class as identified by the ANN

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	50	6	0	0	0	0	0	0	0	0
1	0	56	5	0	0	0	0	0	0	0
2	0	0	77	0	0	0	0	0	0	0
3	1	0	0	80	0	1	0	0	0	0
4	0	14	11	0	66	1	0	0	0	0
5	0	0	58	0	0	21	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	2	0	0	0	7	1	0	0	0	62

5.3 Comparison with other Classification Techniques

The two other classification techniques that were studied using the same data set were the *Backpropagation Training Algorithm* and a *Statistical Classification Technique.* The Best results obtained from the Backpropagation neural net was an error rate of 12.1% with four features f0, f1,f2 and f3. Three layers, made up from an input layer with 4 units, a hidden layer with 16 units and an output layer with 10 units constituted the architecture of the net. The amount of training required for this algorithm was very large. This classification obtained was from several tests that were performed using a different number of hidden units each time for the same set of four features.

The statistical method used was the **resubstitution method using the standard $(D)^2$ distance measure.** The best results for some of the feature sets are summarized in the table below (Table 2) [*Gayvert89*]

Feature Set	Percentage Accuracy
F0, F2	50
F1 - F3	84.2
F1, F2	72.1
F0 - F3	93
F0-MF3	98.5
F0-MF2	98.5
F0-MF1	95.5
F1, F2, MF1, MF2	91
F1-MF3	97.5
F0-F2, MF1	95
F0-F2, MF1	95
F0-F2, MF0-MF3	98

Table 2: Results from statistical classification of the Peterson and Barney data

As the above table shows, the statistical classification technique may be considered to be the most accurate. One of the main reasons for this is that it gives more weight to

a particular feature that helps separate the classes, which is not done in the case of the Kohonen Self-Organizing Feature Map.

5.4 Future Extensions and Conclusions

This thesis has researched a small subset of the phoneme recognition problem. There are several extensions to it that could be worked on. The neural net could be modified to be able to accept any feature set that the user wishes to have instead of having to have all required features in a separate data file. The testing and training of the neural net could be carried out for other feature sets, particularly spectral values and cepstral coefficients. It could also be extended to the entire phoneme set. This should make testing possible with an utterance (such as a word) to see movement in the grid as different nodes get activated for the phonemes that make up the utterance. The first few passes of training significantly reduces the error rate after which the improvement is very slow. The accuracy does not improve appreciably even after extensive training. To enhance the performance, a Hidden Markov Model (HMM) or a Backpropagation net could be used on top of the Kohonen net and could be made to use for its input, the output from the Kohonen net. The HMM may use the output of the Kohonen net as the probabilities it needs to carry out the classification. The net topology can be changed to that of a Hexagonal planar lattice (hexagonal neighborhood) as shown in Figure 26 below, to see what effect the shape of the grid has on the classification.

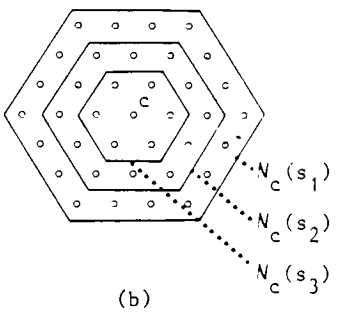


Figure 26: Hexagonal Topology (*adapted from [Kohonen88]*)

The two parts of the graphical user interface can be combined into one and run on a machine whose response is better than what is available now. The addition of a color palette would give the user the ability to choose colors to represent the vowels, foreground and background. A cut and paste facility, if provided, would

also enable the user to visualize the wrap around effect better. A pie-chart or a torus shaped output layer would also help visualize the formation of the maps better.

In conclusion, it may be said of Kohonen's Self-Organizing Feature Map that it performs moderately well. The behavior it exhibits and the misclassifications that are observed are logical. The tests that were performed using a larger grid size and neighborhood (16x16 grid and an initial neighborhood of 4), using features sets that yielded best results with a 8x8 grid, indicate a considerable improvement in the classification accuracy (although this would require more computing resources). An important characteristic that is noticeable from the tabulated results is the increase in the accuracy after one training pass over the data file. The graphical user-interface does help visualize the formation of the maps and the self-organizing process. The movement of the vowels over the map can be observed. Phonotopically similar nodes appear together in the net. There is no confusion among vowels that are unlike. The wrap around effect can also be visually observed during formation of the maps. The addition of a Hidden Markov Model would definitely improve performance and make a reduction in the error associated with the classification.

More research, training and testing needs to be carried out to extend this work to a full phoneme set and make recognition of words using this approach realizable.

6.0 Glossary

* **Backprop Algorithm:** A supervised learning algorithm that uses a feedback gradient descent approach to train a neural network.
* **HMM**: Hidden Markov Model
* **FFT**: Fast Fourier Transform
* **GUI:** Graphical User Interface
* **LPC**: Linear Predictive Coding
* **P&B data set**: Peterson and Barney data set
* **Phonotopic Maps**: A two dimensional map formed by self-organization that directly displays the similarity relations of phonological units.
* **Unsupervised Learning**: A learning mechanism that uses sel-organization and no teaching for neural nets.
* **Voronoi Tessellation**: A concept useful for the illustration of vector space methods for pattern recognition and neural networks. It is a kind of vector quantization where an n-dimensional Euclidean space is partitioned into regions bordered by hyperplanes. These planes constitute the tessellation.
* **XView:** A high level toolkit that conforms to the OpenLook standard for developing GUI's.

7.0 Bibliography

[Cutolo89] Cutolo, F.M., Character Recognition using Kohonen's Neural Network, Master's Project, Graduate Computer Science Department, RIT , January 1989.

[Gayvert89] Gayvert, R.,T., A Statistical Approach to Formant Tracking, unpublished Masters Thesis, Rochester Institute of Technology, 1989.

[Goldstein76] Goldstein, U., Speaker-identifying Features Based on Formant Tracks, JASA, 1976, Vol 59, no. 1, 176-182.

[Hunt75] Hunt, R., K., Berman, N., Journal of Comp. Neurology, 1975, Vol 162, page 43.

[Kangas90] Kangas, J.A., Kohonen, T.K., Laaksonen, J.T., Variants of Self-Organizing Maps, IEEE Transactions on Neural Networks, March 1990, Vol. 1, No. 1, 93-99.

[Kohonen82] Kohonen, T., Clustering, Taxonomy, and Topological Maps of Patterns, In proceedings of the Sixth International Conference on Pattern Recognition, 1982, 114 -128.

[Kohonen82] Kohonen, T., Self-organized formation of topologically correct feature maps, Biological Cybernetics, 1982, 43:56-69.

[Kohonen84] Kohonen, T.K., Makisara, K., Saramaki, T., Phonotopic Maps- Insightful Representation of Phonological Features for Speech Recognition, In proceedings of the Seventh International Conference on Pattern Recognition, 1984, 182-85.

[Kohonen88] Kohonen, T.K., The "Neural" Phonetic Typewriter, IEEE Computer Magazine, March 1988, 11-22.

[Kohonen88a] Self-Organization and Associative Memory, Third Edition, Springer-Verlag Series, 1988.

[Lippmann87] Lippmann, R.P., An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987, 4-22.

[Lippmann88] Lippmann, R.P., Neural Network Classifiers for Speech Recognition, The Lincoln Laboratory Journal, 1988, Vol. 1, No. 1, 107-124.

[Melton90] Melton, M., Phan, T., Reeves, D., Van den Bout, D., VLSI Implementation of TInMANN, NIPS Conference, 1990.

[Minifie73] Minifie, F., Normal Aspects of Speech, Hearing and Language. New Jersey, Prentice-Hall, 1973.

[Olton77] Olton, D., S., Scientific American Vol 236, 82 - 106, 1977.

[Parsons86] Parsons, T., Voice and Speech Processing, New York, McGraw-Hiill, 1986.

[Peterson52] Peterson, G., Barney, H., Control Methods used in a Study of the Vowels, Journal of the Acoustics Society, Vol 24, 175-184, 1952.

[Picket80] Picket, J.,M., The Sounds of Speech Communication, Baltimore, University Park Press, 1980.

[Stam89] Stam, D., C., Vowel Recognition in Continuous Speech, Masters Thesis, Rochester Institute of Technology, 1989.

[Stevens63] Stevens, K., A., House, A., S., Perturbation of Vowel Articulations by Consonantal Context: An Acoustical study, Journal of Speech, Hearing and Resp., 1963, Vol 6(2), 111-128.

[Snyder90] Snyder, W., Nissman, D., Van den Bout, D., Bilbro, G., Kohonen Networks and Clustering, NIPS Conference 1990.

[Wilder75] Wilder, L., Articulatory and Acoustic Characteristics of Speech Sounds, Understanding Language, Massaro, D (ed) , New York, Academic Press, 1975.

8.0 Appendix

```
/*************************************************************************/
/* This is the Program for vowel recognition using artificial neural networks */
/* The Neural Net has been implemented using the Kohonen net algorithm      */
/* This is the main function that is being used for the neural net in the   */
/* absence of the GUI and Xview.This function reads in the initial values for */
/* parameters, checks for filenames and the presence of the files, assigns  */
/* default values & once all the parameters are in, passes control over to the*/
/* kohonen function.
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"


/* declare the arrarys  that are used for the algorithm                    */
/* wt[i][j][k] means input node i to output(j,k) in the output layer       */
/* distance[i][j] refers to distance between node i and node j in the output*/
/* layer.                                                                  */
/* state[i] refers to i(th) node in input layer                            */
/* layers is the array that stores the number of nodes in the neural net,  */
/* the x_dimension of the output layer and the y_dimension of the output layer*/


    double ***wt;
    double *state;
    double **distance;
    double **local_dist;
    int ***table;
    int ***check;


main(argc,argv)
int argc;
char **argv;
{

        int c;
        int w_flag_presence =0;
        int neighborhood = 0;
        int final_neighborhood = 0;
        int initial_neighborhood = 0;
        int read_neighborhood = 0;
        int read_final_neighborhood = 0;
        int total_iterations = 0;
        double control_neighborhood = 0.0;
        int x_dimension_output_layer = 0;
        int y_dimension_output_layer = 0;
        int pass_num = 0;
        int data_samples = 0;
        int total_passes =0;
```

```
        int input_layer =0;
        double alpha =0.0;
        double delta = 0.0;
        double decay = 0.0;
        float f_decay = 0.0;
        int how_often =0;
       int *layers;
        /*char *answer;*/
        int answer;
        int rturn;
        char confuse_file[40];
        char input_wts_filename[40];
        char output_wts_filename[40];
        char data_filename[40];
        char result_filename[40];
        int i,flag;

        FILE *fd_input_wt,*fd_output_wt,*fd_data,*fd_result,*fd_confuse,*fopen();

/*
Get the input
*/

        printf("\nWould you like to start with fresh weights....?(y/n)");
        answer = getchar();
        rturn =getchar();

       if(answer == 'y')
        {
        printf("\nEnter the number of nodes in the input layer....");
        scanf("%d",&input_layer);

        printf("\nEnter the number of nodes in the X-dimension....");
        scanf("%d",&x_dimension_output_layer);

        printf("\nEnter the number of nodes in the Y-dimension....");
        scanf("%d",&y_dimension_output_layer);

        printf("\nEnter the initial neighborhood ....");
        scanf("%d",&initial_neighborhood);

        printf("\nEnter the final neighborhood ....");
        scanf("%d",&final_neighborhood);

        rturn =getchar();
        }

        if(answer == 'n')
        {
        printf("\nEnter the input weights filename....");
        gets(input_wts_filename);
        }
        printf("\nEnter the output weights filename....");
```

```
        gets(output_wts_filename);

        printf("\nEnter the data filename....");
        gets(data_filename );


      if(answer == 'y')
        {
        printf("\nEnter the alpha value \(between 0.1 and 0.99\) ");
        scanf("%le",&alpha);

        printf("\nEnter the how often value (how often you would like to see results printed ....");
        scanf("%d",&how_often);

        printf("\nEnter the number of passes over the data file ....");
        scanf("%d",&total_passes );

        printf("\nEnter the number of data samples in the data file ....");
        scanf("%d",&data_samples );
      }

/*********************************************************************/

 /* check for case when when you start with fresh weights */

 if(answer=='y')
  {
    flag = Initialize; /* set flag for initializing weights */


  /* check for default values and assign if necessary */


    if(initial_neighborhood ==0)
        initial_neighborhood = Default_neighborhood;

    if(final_neighborhood ==0)
        final_neighborhood = Default_final_neighborhood;

    if(input_layer ==0)
        input_layer = Default_input_layer_nodes;

    if(x_dimension_output_layer ==0)
        x_dimension_output_layer = Default_x_dimension_output_layer;

    if(y_dimension_output_layer ==0)
        y_dimension_output_layer = Default_y_dimension_output_layer;


    if(total_passes ==0)
      total_passes = Default_total_passes;
```

```
  if(how_often ==0)
    how_often = Default_how_often;

  if(data_samples ==0)
    data_samples = Default_data_samples;

  /* malloc for layers */

   if(!(layers = (int *) malloc(Three * sizeof(int))))
     {
      fprintf(stderr,"\nOut of memory\n");
      exit(2);
     }

        fprintf(stderr,"kohonen %d\t",input_layer);
        fprintf(stderr,"kohonen %d\t",x_dimension_output_layer);
        fprintf(stderr,"kohonen %d\t",y_dimension_output_layer);

     layers[Input]      = input_layer;
     layers[Output_X]   = x_dimension_output_layer;
     layers[Output_Y]   = y_dimension_output_layer;

 }

/* check for case when  you start with old weights  */


else if(answer=='n')
 {


   if(strcmp(input_wts_filename,"") <= 0)
    {
     fprintf(stderr,"\nInput filename Mandatory\n");
     fprintf(stderr,"when -wn  option used \n");
     fprintf(stderr,"Use filename with -i option \n\n");
     exit(2);
    }

   if((fd_input_wt = fopen(input_wts_filename,"r")) == NULL)
    {
     printf("\nFile %s can't be opened or doesn't exist\n",input_wts_filename);
     exit(2);
    }

/* set flag ,read the number of layers,do a malloc & then read dimensions*/

   flag = Read_weight;
   rewind(fd_input_wt);

   if(!(layers = (int *) malloc(Three* sizeof(int))))
     {
      fprintf(stderr,"\nOut of memory\n");
```

```
    exit(2);
   }

 for(i=0;i<Three;i++)
  {
  fscanf(fd_input_wt,"%d",&layers[i]);
  }

   fscanf(fd_input_wt,"%d",&initial_neighborhood);
   fscanf(fd_input_wt,"%d",&final_neighborhood);

}

/*****************************************************************/


/* check for filename presence & open them */


  if(strcmp(data_filename,"") <= 0)
   {
    fprintf(stderr,"\n  Data filename Mandatory\n");
    fprintf(stderr,"\n  Use -d option\n");
    exit(2);
   }

   if((fd_data = fopen(data_filename,"r")) == NULL)
     {
      printf("\n File %s can't be opened or does not exist\n",data_filename);
      exit(2);
     }


  if(strcmp(output_wts_filename,"") >0)
   {
    if((fd_output_wt = fopen(output_wts_filename,"w")) == NULL)
      {
       printf("\n File %s can't be opened \n",output_wts_filename);
       exit(2);
      }
   }
  else
   {
    fprintf(stderr,"\n Output filename not specified\n");
    fprintf(stderr,"\n Weights will be written to %s\n\n",Default_output_wts_file);
    if((fd_output_wt = fopen(Default_output_wts_file,"w")) == NULL)
     {
      printf("\n File %s can't be opened \n",Default_output_wts_file);
      exit(2);
     }
   }
```

```
/*****************************************************************/

/* dynamically allocate memory for the arrays */

        mem_alloc(layers);


/*****************************************************************/

/*  initialize or read values depending on the flag values */

   if(flag == Initialize)
    {
     initialize(layers);
    }

    else if(flag == Read_weight)
     {
     read_initial_wts(fd_input_wt,fd_result,layers,&alpha,&delta,&control_neighborhood,&decay,
&how_often,&data_samples,&pass_num,&total_passes);

       fprintf(stderr,"The initial neighborhood was %d\n",initial_neighborhood);
       fprintf(stderr,"The final neighborhood was  %d\n",final_neighborhood);
       fprintf(stderr,"The alpha value was  %e\n",alpha);
       fprintf(stderr,"The delta value was  %e\n",delta);
       fprintf(stderr,"The control neighborhood value was  %e\n",control_neighborhood);
       fprintf(stderr,"The decay value was  %e\n",decay);
       fprintf(stderr,"The testing was being done once every %d pass(es)\n",how_often);
       fprintf(stderr,"The number of data samples were  %d\n",data_samples);
       fprintf(stderr,"The current pass number was  %d\n",pass_num);
       fprintf(stderr,"The total number of passes were %d times the dimensions the net\n\n",total_
       fprintf(stderr,"\n Would you like to use new values (y/n)\t");

         c = getchar();

         if(c=='y')
         {
          control_neighborhood =0.0;
          get_input(&initial_neighborhood,&final_neighborhood,&alpha,&delta,
&how_often,&data_samples, &pass_num, &total_passes);

         }

      }

  /*****************************************************************/


  /* pass control to the Kohonen function */


kohonen(fd_data,fd_output_wt,fd_result,layers,initial_neighborhood,final_neighborhood,alpha,
delta,control_neighborhood,decay,how_often,data_samples,pass_num,total_passes);
```

```
 /* close all files */

      fclose(fd_input_wt);
      fclose(fd_output_wt);
      fclose(fd_data);

      printf("kohonen ENDS.. \t ");


}    /* end main */


get_input(initial_neighborhood,final_neighborhood,alpha,delta,how_often,data_samples,
pass_num,total_passes)
int *initial_neighborhood, *final_neighborhood;
int *how_often, *data_samples, *pass_num, *total_passes;
double *alpha;
double *delta;
{

float f_alpha,f_delta;

        printf("\nEnter the initial neighborhood ..(n)..");
        scanf("%d",&(*initial_neighborhood));
        printf("\n neigh ...%d",*initial_neighborhood);

        printf("\nEnter the final neighborhood ..(n)..");
        scanf("%d",&(*final_neighborhood));
        printf("\n neigh ...%d",*final_neighborhood);

        printf("\nEnter the alpha value \(between 0.1 and 0.01\) ");
        scanf("%le",&(*alpha) );
        fprintf(stderr,"\n alpha ...%e",*alpha);

        printf("\nEnter the delta value (between 0.95 and 0.995) ..");
        scanf("%le",&(*delta) );
        fprintf(stderr,"\n delta ...%e ",*delta);

        printf("\nEnter the how often value ..(n)..");
        scanf("%d",&(*how_often));
        printf("\n how_often passes ...%d",*how_often);

        printf("\nEnter the initial pass number ..(n)..");
        scanf("%d",&(*pass_num ));
        printf("\n training passes ...%d",*pass_num);

        printf("\nEnter the number of passes over the data file ..(n)..");
        scanf("%d",&(*total_passes ));
        printf("\n training passes ...%d",*total_passes);

        printf("\nEnter the number of data samples in the data file ..(n)..");
        scanf("%d",&(*data_samples));
}
```

```
/*************************************************************************/
/* This is the function that initializes the weights when the network is */
/* first used. The function drand48 is the random number generator.      */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

extern          double ***wt;
double     drand48();

initialize(layers)
int  *layers;
{

int input_node,output_node_x,output_node_y;
double dummy =0.0;
long seed;

seed = rand();

srand48(seed);

  for(input_node=Zero;input_node<layers[Input];input_node++)
  {
   for(output_node_x=Zero;output_node_x<layers[Output_X];output_node_x++)
   {

   for(output_node_y=Zero;output_node_y<layers[Output_Y];output_node_y++)
    {

     wt[input_node][output_node_x][output_node_y] = drand48();

    }

   }

 }

} /* end of initialize */
```

```
/*************************************************************************/
/* This is the function that allocates memory dynamically to the arrays  */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     double **local_dist;
 extern     int ***table;
 extern     int ***check;

mem_alloc(layers)
int  *layers;
{
int r,l,i,j,k,m;

/************** memory for the wt array ***********************/

 if(!(wt = (double ***) malloc(layers[Input] * sizeof(double**))))
  {
   fprintf(stderr,"\n Out of memory or Allocation failure \n");
   exit(2);
  }

 for(k=0;k<layers[Input];k++)
  {
    if(!(wt[k] = (double**) malloc(layers[Output_X]*sizeof(double*))))
     {
      fprintf(stderr,"\n Out of memory or Allocation failure \n");
      exit(2);
     }


    for(m=0;m<layers[Output_X];m++)
     {
      if(!(wt[k][m] = (double *) malloc(layers[Output_Y] *sizeof(double))))
      {
       fprintf(stderr,"\n Out of memory or Allocation failure \n");
       exit(2);
      }
     }
  }


/************** memory for the check array *****************/

 if(!(check = (int ***) malloc(10 * sizeof(int**))))
```

```
 {
  fprintf(stderr,"\n Out of memory or Allocation failure \n");
  exit(2);
 }

for(k=0;k<10;k++)
 {
   if(!(check[k] = (int **) malloc(layers[Output_X]*sizeof(int*))))
    {
     fprintf(stderr,"\n Out of memory or Allocation failure \n");
     exit(2);
    }


   for(m=0;m<layers[Output_X];m++)
    {
     if(!(check[k][m] = (int *) malloc(layers[Output_Y] *sizeof(int))))
     {
      fprintf(stderr,"\n Out of memory or Allocation failure \n");
      exit(2);
       }
    }
  }


/************** memory for the table array *****************/
 if(!(table = (int ***) malloc(10 * sizeof(int**))))
  {
   fprintf(stderr,"\n Out of memory or Allocation failure \n");
   exit(2);
  }

 for(k=0;k<10;k++)
  {
    if(!(table[k] = (int **) malloc(layers[Output_X]*sizeof(int*))))
     {
      fprintf(stderr,"\n Out of memory or Allocation failure \n");
      exit(2);
     }


    for(m=0;m<layers[Output_X];m++)
     {
      if(!(table[k][m] = (int *) malloc(layers[Output_Y] *sizeof(int))))
      {
       fprintf(stderr,"\n Out of memory or Allocation failure \n");
       exit(2);
        }
     }
   }

/************** memory for the distance array *****************/
```

```
 if(!(distance = (double **) malloc(layers[Output_X] * sizeof(double *))))
   {
    fprintf(stderr,"\n Out of memory or Allocation failure \n");
    exit(2);
   }
 for (i=0;i<layers[Output_X];i++)
  {
   if(!(distance[i] = (double *) malloc(layers[Output_Y] * sizeof(double))))
   {
    fprintf(stderr,"\n Out of memory or Allocation failure \n");
    exit(2);
   }
  }

/************** memory for the state array ********************/

if(!(state = (double *) malloc(layers[Input] * sizeof(double))))
   {
    fprintf(stderr,"\n Out of memory or Allocation failure \n");
    exit(2);
   }

/************** memory for the local dist array ********************/

 if(!(local_dist= (double **) malloc( (2*layers[Output_X] +1) * sizeof(double *))))
   {
    fprintf(stderr,"\n Out of memory or Allocation failure \n");
    exit(2);
   }
 for (i=0;i<layers[Output_X];i++)
  {
   if(!(local_dist[i] = (double *) malloc( (2 * layers[Output_Y] +1) * sizeof(double))))
   {
    fprintf(stderr,"\n Out of memory or Allocation failure \n");
    exit(2);
   }
  }
/*****************************************************************/

}
/* end of mem_alloc */
```

```
/*****************************************************************************/
/* This is the function that runs the kohonen algorithm. The main routine    */
/* passes control for the required number of passes to be executed. This     */
/* calls the other functions to do the tasks of reading data,calculating     */
/* ouptut, adjusting weights, printing the confusion matrix, writing to the  */
/* file the weights, printing out the tables                                 */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <values.h>
#include "kohonen.h"


extern          double ***wt;
extern    double *state;
extern    double **distance;
extern    double **local_dist;
extern    int ***table;
extern    int ***check;

kohonen(fd_data,fd_output_wt,fd_result,layers,initial_neighborhood,final_neighborhood,alpha,
delta, control_neighborhood, decay, how_often, data_samples, passes, total_passes)

FILE *fd_data,*fd_output_wt,*fd_result;
int *layers,initial_neighborhood,final_neighborhood;
int passes,total_passes,how_often, data_samples;
double delta, control_neighborhood, decay;
double alpha;
{

        int s_num =0;
        int pass_num =0;
        int outer_loop_control =0;
        int start_pass =0;
        int c_often =0;
        int t_passes =0;
        int i,j,k,l =0;
        int vowel_type=0;
        int vowel_num=0;
        int vowel_class=0;
        int min_out_x =0;
        int min_out_y =0;
        int row =0 ;
        int column =0 ;
        int neighborhood = 0;
        int total_iterations =0;
        double c_delta =0.0;
        double c_alpha =0.0;
        double c_decay=0.0;
        double c_neighborhood =0.0;
        int start_num =0;
```

```
        int input_node =0;
        int input_da[1520][2];
        double input_db[1520][8];


/*
This determines the number of passes for the training
*/
        outer_loop_control =layers[Output_X]*layers[Output_Y] * total_passes;

        total_iterations =outer_loop_control * data_samples ;

        c_alpha = alpha;
        c_delta = delta;
        t_passes = total_passes;
        c_often = how_often;
       start_pass = passes;
        c_decay = decay;
        c_neighborhood = control_neighborhood;

/*
This is done if the training is being started from the beginning or if the
value of the neighborhoods has been changed when the training was restarted
froma previous run
*/
        if(c_neighborhood ==0.0)
        {
        c_decay =exp(log((float)final_neighborhood/(float)initial_neighborhood)
                                                    /(float)total_iterations);
        c_neighborhood= (double)initial_neighborhood;


        c_delta = exp(log( 0.005 /alpha) / total_iterations);

/*
The final value of alpha can be varied. The values that I have used are
0.0000001 and 0.005.
The formula below was used initially but delta was too small to affect the weights
        c_delta = exp(log( sqrt(MINDOUBLE) /alpha) / total_iterations);
*/

         }

/*
This is the variation where the data was all read in in the beginning
In this case all 1520 data sets and 8 features
*/

        for(i=0; i<1520;i++)
        {
         fscanf(fd_data, "%d %d ", &input_da[i][0], &input_da[i][1]);

          for(j=0;j<8;j++)
```

```
        {
         fscanf(fd_data, "%le", &input_db[i][j]);
        }
       }

/*
Start the training
*/

      for(pass_num=start_pass;pass_num <outer_loop_control;pass_num++)
      {

        for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
        {
            s_num =      input_da[vowel_num][0];
          vowel_class =  input_da[vowel_num][1];


              for(input_node=Zero;input_node<layers[Input];input_node++)
                  {
                   state[input_node] =input_db[vowel_num][input_node +1];
                  }

         calculate_distance(fd_result, layers,&min_out_x,&min_out_y);


         adjust_wt(fd_result,layers,&c_alpha,&c_neighborhood,&min_out_x,&min_out_y);
        c_neighborhood = c_neighborhood * c_decay ;
         c_alpha = c_alpha * c_delta;


       } /* for inner loop ends here */

/*
********************* TESTING ***************************
*/

       if((pass_num % how_often) == 0)
        {
          fprintf(stderr,"control  %d\n",pass_num);

           for(vowel_type=0;vowel_type<Vowel_type_max;vowel_type++)
            {
             for(row =Zero;row <layers[Output_X];row++)
              {
               for(column=Zero;column<layers[Output_Y];column++)
                {

                   table[vowel_type][row][column] = 0;
                }
              }
            }
```

```
            for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
             {

                    s_num = input_da[vowel_num][0];
                    vowel_class = input_da[vowel_num][1];

                    for(input_node=Zero;input_node<layers[Input];input_node++)
                    {
                      state[input_node] =input_db[vowel_num][input_node +1];
                    }


                    calculate_distance(fd_result, layers,&min_out_x,&min_out_y);
                    table[vowel_class][min_out_x][min_out_y]++;
                }
           map_to_table(&pass_num,&c_often,layers);

         write_final_wts(fd_output_wt,layers,&initial_neighborhood,&final_neighborhood,
&c_alpha,&c_delta,&c_neighborhood,&c_decay,&how_often, &data_samples, &pass_num,
&total_passes);

           fflush(fd_output_wt);
           }

           if(c_alpha <= 0.0)
            {
             fprintf(stderr,"\t panic ..pass num= %d alpha =  %e\t",pass_num,c_alpha);
             }

       }  /* end of for  outer loop  */

       write_final_wts(fd_output_wt,layers,&initial_neighborhood,&final_neighborhood,
&c_alpha,&c_delta,&c_neighborhood,&c_decay,&how_often, &data_samples, &pass_num,
&total_passes);

}
  /* end of control */
```

```
/*************************************************************************/
/* This is the function for reading data from the file. The data is read */
/* one feature set at a time.The data file is assumed to have the following */
/*: serial number,class number of the vowel, & the different features for */
/* that vowel. A few implementations of my neural net did not have this */
/* function because I had to reduce the number of disk accesses. In those */
/* cases the data was read in the control function and stored in an array */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     int ***table;
 extern     int ***check;

read_input(fd_data,fd_result,layers,vowel_class)
FILE *fd_data, *fd_result;
int *layers,*vowel_class;

{
int input_node=0;
double dummy1,dummy2=0.0;

#ifdef DEBUG
        fprintf(fd_result,"read_input  %d\n",layers[0]);
        fprintf(fd_result,"read_input  %d\n",layers[1]);
        fprintf(fd_result,"read_input  %d\n",layers[2]);
#endif

        fscanf(fd_data,"%d",&(*vowel_class));

/*
Read the various feature values
It is assumed that the data file has as many features as is required
*/

 for(input_node=Zero;input_node<layers[Input];input_node++)
   {
    fscanf(fd_data,"%le",&state[input_node]);
/*
The state array has the feature values stored in it
*/


   }

}
/* end of read data */
```

```
/*****************************************************************************/
/* This is the function that calculates the distances between the input vector*/
/* and the weights for the connections in the network                         */
/* The Eucledian distance measure is used here. This function returns the x & */
/* y coordinates of the best node (i.e., the node with the minimum distance)  */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

/* extern arrays */

 extern          double ***wt;
 extern    double *state;
 extern    double **distance;
 extern    int ***table;
 extern    int ***check;

calculate_distance(fd_result,layers,min_out_x,min_out_y)
FILE *fd_result;
int *layers,*min_out_x,*min_out_y;
{
    int input_node,output_node_x,output_node_y;
    double temp_dist =0.0;
    double xx=0.0;
    double minimum_value=0.0;

    minimum_value = 1000000.0;
    *min_out_x = -1;
    *min_out_y = -1;

#ifdef DEBUG
    fprintf(fd_result,"calc_distance %d\n",layers[Output_X]);
    fprintf(fd_result,"%d\n",layers[Output_Y]);
    fprintf(fd_result,"%d\n",layers[Input]);
    fprintf(fd_result,"%e\n",state[0]);
    fprintf(fd_result,"%e\n",state[1]);
    fprintf(fd_result,"%e\n",state[2]);
    fprintf(fd_result,"%e\n",state[3]);
#endif


/*
The three for loops are for the X-dimension of the net, the y-dimension of the net
and the number of input nodes in the input layer respectively.
*/

    for(output_node_x=Zero;output_node_x<layers[Output_X];output_node_x++)

     {
      for(output_node_y=Zero;output_node_y<layers[Output_Y];output_node_y++)
```

```
        {
          distance[output_node_x][output_node_y] = 0.0;
         xx=0.0;
          temp_dist=0.0;

          for(input_node=Zero;input_node<layers[Input];input_node++)
          {

/*
This is the loop where the distances are computed (Eucledian)
*/

            xx =  state[input_node] - wt[input_node][output_node_x][output_node_y];

            distance[output_node_x][output_node_y] += xx*xx;

/*
I just broke it up to avoid typing it all in one line
*/

           }

          temp_dist = distance[output_node_x][output_node_y];

/*
This sqrt is not necessary but has been used nonetheless
*/

          distance[output_node_x][output_node_y] = sqrt(temp_dist) ;

          if(distance[output_node_x][output_node_y] <minimum_value)
            {
/*
Get the best node's x and y coordinates and the distance
*/

              minimum_value = distance[output_node_x][output_node_y];
              *min_out_x = output_node_x;
              *min_out_y = output_node_y;

/*
minout_x and min_out_y are the coordinates of the node that has the minimum
distance
*/
            }


        }

    }

} /* end of function */
```

```
/*************************************************************************/
/* This is the function for adjusting wts. The adjustment is based on the    */
/* alpha value and the proximity of the node to the winning node (node with) */
/* least distance. The distance that is  used for  determining the proximity */
/* is the  Eucledian distance measure. The factor ratio is used to determine */
/* the amount of the update. Closer the node to the winning node, more is the*/
/* change in the weight. A check feature is used to ensure that all nodes in */
/* the grid in that adjustment cycle are updated only once. This function uses*/
/* WRAP AROUND                                                               */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     double **local_dist;
 extern     int ***table;
 extern     int ***check;

adjust_wt(fd_result,layers,alpha,neighborhood,min_out_x,min_out_y)
FILE  *fd_result;
double *alpha;
int *layers,*min_out_x,*min_out_y;
double *neighborhood;
{
int x = 0;
int y =0;
int x1=0;
int y1=0;
int upper_x = 0;
int upper_y = 0;
int lower_x = 0;
int lower_y = 0;
int input = 0;
int i=0;
int j=0;
int x_row =0;
int y_row =0;
int adjust_neighborhood;
int neighborhood_limit = 0;
double local_alpha = 0.0;
double dist[16][16];
double ratio=0.0;

/*
The nint function is available only on the Suns. If it is not available,
use the following
*/
```

```
  if( *neighborhood >= (int) *neighborhood+ 0.5 )
    adjust_neighborhood= (int) *neighborhood +1;
  else
    adjust_neighborhood= (int) *neighborhood ;

  neighborhood_limit = 2 * ( adjust_neighborhood ) + 1;


/*
Malloc for the dist array. Use this if you wish to after changing the declaration
above

  if(!(dist = (double **) malloc(neighborhood_limit * sizeof(double)) ))
   {
    fprintf(stderr, "\n Out of memory or Allocation failure\n");
    exit(2);
  }
  for(i=0;i<neighborhood_limit;i++)
  {
  if(!(dist[i] = (double *) malloc(neighborhood_limit * sizeof(double) )))
   {
    fprintf(stderr, "\n Out of memory or Allocation failure\n");
    exit(2);
   }
  }
 */


/*
Set all the elements in the array to 0
*/

  for(i=0 ; i<neighborhood_limit;i++)
   {
     for(j=0;j<neighborhood_limit;j++)
      {
        dist[i][j]= 0.0;
      }
   }

/*
The check array keeps track of the weight update and ensures that each
node gets updated only once.
*/

  for(x_row = Zero; x_row<layers[Output_X];x_row++)
  {
   for(y_row=0;y_row<layers[Output_Y];y_row++)
    {
     for(input=Zero;input <layers[Input];input++)
     {
        check[input][x_row][y_row]= 0;
     }
```

```
    }
  }

/*
Compute the distances between the nodes
*/

  for(i=0 ; i<neighborhood_limit;i++)
   {
   for(j=0;j<neighborhood_limit;j++)
    {
        dist[i][j]= sqrt( pow(  ( *neighborhood - (double) i), 2.0)  +
                          pow(  ( *neighborhood - (double) j), 2.0  ) );

    }
   }


/*
Compute the bounds for the update
*/

  local_alpha = *alpha;
  lower_x     = *min_out_x - adjust_neighborhood;
  lower_y     = *min_out_y - adjust_neighborhood;
  upper_x     = *min_out_x + adjust_neighborhood;
  upper_y     = *min_out_y + adjust_neighborhood;

  for(x=lower_x, i=0;x<=upper_x;x++,i++)
  {


/*
This is where the wrap around takes place. The coordinates x1,y1
determine the final node to be used for the weight update
*/

     if(x <0)
       x1 = x + layers[Output_X] ;

     else  if(x > (layers[Output_X] -1))
       x1 = x % layers[Output_X] ;

     else
       x1 = x;

   for(y = lower_y, j=0;y<= upper_y;y++,j++)
    {

     if(y <0)
       y1 = y + layers[Output_Y];
```

```
    else  if(y > (layers[Output_Y] - 1 ) )
      y1 =  y % layers[Output_Y];

    else
      y1 = y;

/*
Compute the ratio as explained in the text. if the distance between the
nodes is greater than the current neighborhood saturate the ratio at 1
Use the weight adjust formula that incorporates the ratio
*/

    for(input=Zero;input <layers[Input];input++)
    {
      if( dist[i][j] > *neighborhood)
        ratio = 1.0;
      else
        ratio = dist[i][j]/ *neighborhood;


     if(check[input][x1][y1] ==0)
        {
        wt[input][x1][y1] = wt[input][x1][y1] +
              (1.0 - ratio) * local_alpha * (state[input] - wt[input][x1][y1]);

       check[input][x1][y1]= 1;

       }

    }

   }  /* end of for for y */

  }  /* end of for for x */


} /* end of adjust weights */
```

```
/*************************************************************************/
/*This is the function that prints out the tables. It also creates a file that*/
/*has the activations of the nodes for that pass. This file can be used for  */
/*drawing the phonotopic maps by the graphical user interface. The primary   */
/*name of the file is the same as that of the weights file. The extension    */
/*however, is .tab (for table). Ten tables are printed out one for each vowel */
/*A summary is also printed out the best nodes for that pass for each vowel  */
/*This is also the means of obtaining output about the training when the GUI  */
/* is not available                                                          */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     double **local_dist;
 extern     int ***table;
 extern     int ***check;

map_to_table(pass_num,how_often,layers)
int *pass_num,*how_often;
int *layers;
{
   int row,column;
   FILE  *out_fd, *new_fd;
   int class_num;
   int x,y,i,j;
   int final_table[20][20];
   int vowel_type;
   int vowel_no;
   int vowel_class;
   int max_times;
   int max_x;
   int max_y;
   char *out_char, *outfile;

   fprintf(stderr,"\nPrinting table for pass number %d \n",*pass_num);
   fprintf(stderr,"once every %d passes \n",*how_often);

/*
Create the file name, open the file and write the information to the file
*/

   out_char = (char *) malloc(6*sizeof(char));
   outfile = (char *) malloc(30 * sizeof(char));
   strcpy(outfile,"");
   strcat(outfile,"s");
   sprintf(out_char,"%d",*pass_num);
   printf("%s\n\n",out_char);
```

```
    strcat(outfile,out_char);
    strcat(outfile,".tab");

    if((new_fd = fopen(outfile,"w")) == NULL)
      {
        fprintf(stderr,"Can't open %s for writing\n",outfile);
        exit(1);
      }

       fprintf(new_fd, "\n");

         for(row =Zero;row <layers[Output_X];row++)
          {
           for(column=Zero;column<layers[Output_Y];column++)
                {
                 for (vowel_type=0; vowel_type<10;vowel_type++)
                 {
                  fprintf(new_fd,"%d ",table[vowel_type][row][column]);

                 }
                 fprintf(new_fd, "\n");
                }
          }
        fflush(new_fd);
        fclose(new_fd);

/*
Find the best node for each vowel
*/
         for(row =Zero;row <layers[Output_X];row++)
          {
           for(column=Zero;column<layers[Output_Y];column++)
                {
                        final_table[row][column] = 0;
                }
          }


  for (vowel_type=0; vowel_type<10;vowel_type++)
   {
        max_times = 0;
         max_x = 0;
         max_y = 0;

        for(row =Zero;row <layers[Output_X];row++)
         {
          for(column=Zero;column<layers[Output_Y];column++)
                {

                 if(table[vowel_type][row][column] > max_times)
                 {
                  max_times = table[vowel_type][row][column];
                  max_x = row;
```

```
                max_y = column;
              }
            }
        }

        fprintf(stderr,"The best node for vowel %d is %d %d and was %d times \n",vowel_type,
max_x, max_y , max_times);

  }  /* end of for vowel_type  */

/*
Print the 10 tables
*/


 fprintf(stderr,"\n ");


 for (vowel_type=0; vowel_type<10;vowel_type++)
  {
    {
      fprintf(stderr,"    ");

      for(i=Zero;i<layers[Output_Y];i++)
        {
         if(i<10)
        {
         fprintf(stderr,"  %d",i);
         }
         else
         {
         fprintf(stderr," %d",i);
         }

      }

        fprintf(stderr,"\n");
       fprintf(stderr,"    ");

      for(i=Zero;i<3*layers[Output_Y];i++)
        {
         fprintf(stderr,"-");
        }

      for(row=Zero;row<layers[Output_X];row++)
       {
         fprintf(stderr,"\n");
        if( row <10)
         fprintf(stderr,"  %d ",row);
         else
         fprintf(stderr," %d ",row);
```

```
        for(column=Zero;column<layers[Output_Y];column++)
          {
           if(table[vowel_type][row][column] ==Zero)
              {
               fprintf(stderr,"|  ");
              }
           else if(table[vowel_type][row][column] >9)
              {
               fprintf(stderr,"|%d",table[vowel_type][row][column]);
              }
           else if(table[vowel_type][row][column] <=9)
             {
               fprintf(stderr,"| %d",table[vowel_type][row][column]);
             }

           else if(table[vowel_type][row][column] >99)
             {
               fprintf(stderr,"|%d",table[vowel_type][row][column]);
             }
          }

          fprintf(stderr,"|");
          fprintf(stderr,"\n");
         fprintf(stderr,"    ");

         for(i=Zero;i<3*layers[Output_Y];i++)
          {
           fprintf(stderr,"-");
          }

       }
           fprintf(stderr,"\n");
           fprintf(stderr,"\n");
    }
  }
           fprintf(stderr,"\n");


}
/* end of function */
```

```
/*************************************************************************/
/* This is the function that prints the confusion matrix. The  frequency  of */
/* its being printed is specified in the how_often variable. The matrix prints*/
/* a 10x10 set representing the actual classes & the classified classes      */
/* Rows specify the correct classes. Columns specify the classes as the Net  */
/* saw it. The percentage error is also printed.                             */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"


extern    int **confusion;
extern    int *layers;

confus(layers,error, data_samples, pass_num)
int *layers,error, data_samples, pass_num;
  {
   int i,row,col;
   float percnt;

       percnt = ((float)error/data_samples) * 100.0;

        printf("\n Confusion matrix for pass number %d \n",pass_num);
       printf(" Total number of input samples  = %d\n",data_samples);
       printf(" Number of errors in this pass %d \n",error);
       printf(" Percentage error %3.1f \n",percnt);
        printf(" Rows specify the actual classes\n");
        printf(" Columns specify the class as identified by the ANN\n");

           printf("\n      ");

           for(i=0;i<layers[Output_X];i++)
           {
              printf("%4d   ",i);
           }
           printf("\n\n");

           for(row=0;row<layers[Output_Y];row++)
           {
             printf("   %d   ",row);

              for(col=0;col<layers[Output_X];col++)
                {
                    printf("%4d   ",confusion[row][col]);
                }
              printf("\n");
           }
           printf("\n");

   }
```

```
/*************************************************************************/
/* This is the function for reading initial weights from a file . The values */
/* that are initially read in from the main program are the dimensions of the*/
/* net, the initial and final neighborhoods. The weights, alpha, delta, decay*/
/* current neighborhood, number of passes, current pass number, how often val*/
/* -ues are read in this function                                          */
/*************************************************************************/

#include "kohonen.h"
#include <stdio.h>
#include <math.h>
#include <string.h>


extern          double ***wt;
extern    double *state;
extern    double **distance;
extern    double **local_dist;
extern    int ***table;
extern    int ***check;

read_initial_wts(fd_input_wt,fd_result,layers,alpha,delta,control_neighborhood,decay,how_often,
data_samples,pass_num,total_passes)
FILE *fd_input_wt, *fd_result;
int *layers, *how_often, *data_samples, *pass_num, *total_passes ;
double *alpha;
double *delta, *decay, *control_neighborhood;
{
int input_node,output_node_x,output_node_y;
int count =0;


/*
read in the weights
*/


  for(output_node_x=Zero;output_node_x<layers[Output_X];output_node_x++)

  {

  for(output_node_y=Zero;output_node_y<layers[Output_Y];output_node_y++)

   { /* ....to each of the output nodes  */

    for(input_node=Zero;input_node<layers[Input];input_node++)

     { /* connections one input node at a time..... */

      fscanf(fd_input_wt,"%le",&wt[input_node][output_node_x][output_node_y]);

    }
```

```
  }

 }


 /* read ALPHA and DELTA */
 fscanf(fd_input_wt,"%le",&(*alpha));
 fscanf(fd_input_wt,"%le",&(*delta));
 fscanf(fd_input_wt,"%le",&(*control_neighborhood));
 fscanf(fd_input_wt,"%le",&(*decay));
 fscanf(fd_input_wt,"%d",&(*how_often));
 fscanf(fd_input_wt,"%d",&(*data_samples));
 fscanf(fd_input_wt,"%d",&(*pass_num));
 fscanf(fd_input_wt,"%d",&(*total_passes));

}

/* end of read weights */
```

```
/*************************************************************************/
/* This is the function for writing the final weights on to the file . The  */
/* name of the file is created at run time.The values written are the number */
/* of number of nodes in each layer, the initial and final neighborhood     */
/* values, the weights, alpha, delta, current neighborhood, decay, total    */
/* number of training passes, total number of data samples, current pass no. */
/* and the  frequency with which these are being printed out (the how often) */
/* variable. The file is rewound prior to the writing.                      */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     double **local_dist;
 extern     int ***table;
 extern     int ***check;

write_final_wts(fd_output_wt,layers,initial_neighborhood,final_neighborhood,alpha,delta,
control_neighborhood,decay,how_often, data_samples,pass_num,total_passes)
FILE *fd_output_wt;
int *layers,*initial_neighborhood, *final_neighborhood;
double *alpha;
double *delta, *control_neighborhood, *decay;
int *how_often, *data_samples,*pass_num,*total_passes;
{
int input_node,output_node_x,output_node_y;
FILE *out_fd;
char *out_char;
char *outfile;


    out_char = (char *) malloc(6*sizeof(char));
    outfile = (char *) malloc(30 * sizeof(char));

/*
The name of the output wts file is of the form p[pass_number].wts
*/

    strcpy(outfile,"");
    strcat(outfile,"p");
    sprintf(out_char,"%d",*pass_num);
    strcat(outfile,out_char);
    strcat(outfile,".wts");

    if((out_fd = fopen(outfile,"w")) == NULL)
      {
       fprintf(stderr,"Can't open %s for writing\n",outfile);
       exit(1);
```

```
    }

/*
Rewind file, number of nodes in each layer
*/

  rewind(out_fd);

#ifdef DEBUG
    fprintf(stderr,"\nwriting to file\t");
    fprintf(stderr,"%d\t",layers[Input]);
    fprintf(stderr,"%d\t",layers[Output_X]);
    fprintf(stderr,"%d\n",layers[Output_Y]);
    fprintf(stderr,"%d\t",*initial_neighborhood);
    fprintf(stderr,"%d\t",*final_neighborhood);
#endif

    fprintf(out_fd,"%d\n",layers[Input]);
    fprintf(out_fd,"%d\n",layers[Output_X]);
    fprintf(out_fd,"%d\n",layers[Output_Y]);
    fprintf(out_fd,"%d\n",*initial_neighborhood);
    fprintf(out_fd,"%d\n",*final_neighborhood);
    fprintf(out_fd,"\n");


/*
Write weights
*/

  fprintf(out_fd,"\n");

   for(output_node_x=Zero;output_node_x<layers[Output_X];output_node_x++)
   {
     for(output_node_y=Zero;output_node_y<layers[Output_Y];output_node_y++)
      {
       for(input_node=Zero;input_node<layers[Input];input_node++)
        {
          fprintf(out_fd,"%e\t",wt[input_node][output_node_x][output_node_y]);
        }
        fprintf(out_fd,"\n");

      }

   }

    fprintf(out_fd,"%.10e\n",*alpha);
    fprintf(out_fd,"%.10e\n",*delta);
    fprintf(out_fd,"%.10e\n",*control_neighborhood);
    fprintf(out_fd,"%.10e\n",*decay);
    fprintf(out_fd,"%d\n",*how_often);
    fprintf(out_fd,"%d\n",*data_samples);
    fprintf(out_fd,"%d\n",*pass_num);
    fprintf(out_fd,"%d\n",*total_passes);
```

```
    fclose(out_fd);
}

/* end of write weights */
```

```
/*************************************************************************/
/* This is the header file that has all the declarations for the constants*/
/* used in all the programs. The name of this file is kohonen.h          */
/*************************************************************************/

#define  Input 0
#define  Output_X 1
#define  Output_Y 2
#define  Zero 0
#define  Three 3
#define  Initialize 10
#define  Read_weight 20
#define  Vowel_type_max 10
#define  Default_alpha 0.05
#define  Default_delta .97250
#define  Default_total_passes 200
#define  Default_neighborhood 4
#define  Default_final_neighborhood 4
#define  Default_data_samples  1520
#define  Default_x_dimension_output_layer 6
#define  Default_y_dimension_output_layer 6
#define  Default_input_layer_nodes 4
#define  Default_how_often 100
#define  Default_output_wts_file "default.wts"
#define  Default_result_file "default.result"
```

```
/*
This is the program that draws the maps on to the screen for each set of data
that is produced during the training of the neural net. The program takes as
input the node activations for one pass of the data file. The output produced
comprises of mainly 6 phonotopic maps that gives the user an idea about the
organization of the vowels and the classification at that point.
*/

/* Include files needed for the Xview functions */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/canvas.h>
#include <xview/cms.h>
#include <xview/svrimage.h>
#include <xview/xv_xrect.h>
#include <xview/scrollbar.h>
#include <xview/notice.h>
#include <xview/screen.h>

/* Icon bitmap info in the include file */

short vowel_icon_bits[] = {
#include "vowel_icon"
};

/* cursor bits used for images of the different colors */

static unsigned short black_data[] ={
#include "/usr/openwin/share/include/images/black.cursor"
};

/* Color Table Indices */

#define GRAY            0
#define MAGENTA         1
#define RED         2
#define ORANGE          3
#define SALMON          4
#define YELLOW          5
#define SPRINGGREEN     6
#define CYAN            7
#define CORNFLOWERBLUE          8
#define BLUE            9
#define MAROON          10
#define WHITE           11
#define BLACK           12


#define     data_filename       "f0_mf3.big"
#define TRUE 1
```

```
#define FALSE 0

/* six of the image */
#define CHIP_WIDTH     16
#define CHIP_HEIGHT    16

/* Vowel info used for displaying notices */

static char *vowel_array[] = {"IY", "IH","EH", "AE", "ER", "AH", "AA", "AO",
                              "UU", "UH"};


static char *color_array[] = {"MAGENTA", "RED", "ORANGE", "SALMON",
                              "YELLOW", "SPRING GREEN", "CYAN",
                              "CORNFLOWER BLUE", "BLUE", "MAROON","WHITE"};

static char *example_array[] = {"HEED", "HID", "HEAD","HAD", "HEARD", "HUD",
                              "HOD", "HAWED", "HOOD", "WHO'D"};

static char *fg_bg_array[] = {"ForeGround", "BackGround"};
static char *fg_bg_color_array[] = {"GRAY", "BLACK"};

GC gc;
unsigned long *colors;

/* Pixmaps used in the program */

Pixmap  sb_NET_gui;
Pixmap  nodes_map;
Pixmap  win_map;
Pixmap  summary_map;
Pixmap  blend_map;
Pixmap  dummy_map;

XID   xid;
Xv_screen screen;

Frame frame;
Panel panel;
Panel_item  panel_vowels, panel_palette, panel_fg_bg,
          panel_pal_fg_bg, panel_toggle;
Canvas canvas;

Display        *display;
Window can_xwin;
Xv_Window can_xvwin;
Xv_singlecolor rgb_values[13];
XColor pixel;
XColor colors_x[13];
Colormap cmap;

FILE *datafile_fd;
```

```
/* make changes to this if you are using different size grids */

int output_x, output_y;
int table[10][8][8];
int max_table[8][8];
int pass_num;

/* Flags */

int just_created = TRUE;
int toggle_flag= FALSE;
int new_toggle_flag= FALSE;
int compress_flag= FALSE;
int color_display_flag = -1;
int planes;

main(argc,argv)
int argc;
char *argv[];

{

        static char stipple_bits[] = { 0xAA, 0xAA, 0x55, 0x55};

        Icon            v_icon;
        Scrollbar       scrollbar_v, scrollbar_h;
        Cms             control_cms, normal_cms;
        Server_image            vowel_image, choice_image;

        Xv_singlecolor          f_colors[13];
        XFontStruct  *font;
        Xv_cmsdata  cms_data;

        XGCValues   gc_val;
        XGCValues   gc_values;

        int i           , row, col, vowel;

/* Function declarations */

        Xv_opaque   vowel_pressed(),vowel_color_pressed();
        Xv_opaque   fg_bg_pressed(),fg_bg_color_pressed();
        void        next_proc(), quit(), toggle_proc();
        void        canvas_repaint_proc(), color_notify();
        int         draw_in_pixmap();

/* Parse the command line for any parameters that are used */

        xv_init(XV_INIT_ARGC_PTR_ARGV,&argc,argv,NULL);

/* Create a server image for the vowel icon */

        vowel_image = (Server_image) xv_create(NULL, SERVER_IMAGE,
```

```
                XV_WIDTH,    64,
                XV_HEIGHT,64,
                SERVER_IMAGE_BITS, vowel_icon_bits,
                NULL);

/* Create a control cms for the colors of the panel items */

        control_cms = xv_create(XV_NULL, CMS,
                CMS_SIZE, 13 + CMS_CONTROL_COLORS,
                CMS_NAMED_COLORS, "gray", "magenta", "red", "orange",
                  "salmon", "yellow", "spring green", "cyan",
                  "cornflower blue", "blue", "maroon", "white","black",  NULL,
                CMS_CONTROL_CMS,   TRUE,
                NULL);

/* Create a server image for panel choice images*/

      choice_image = (Server_image) xv_create(XV_NULL, SERVER_IMAGE,
                                XV_WIDTH, CHIP_WIDTH,
                                XV_HEIGHT, CHIP_HEIGHT,
                                SERVER_IMAGE_DEPTH, 1,
                                SERVER_IMAGE_BITS, black_data,
                                0);

/* Create a cms for the colors of the canvas */

       normal_cms = xv_create(XV_NULL, CMS,
                CMS_SIZE,   13,
                CMS_NAMED_COLORS, "gray", "magenta", "red", "orange",
                  "salmon", "yellow", "spring green", "cyan",
                  "cornflower blue", "blue", "maroon", "white","black",  NULL,
                NULL);

        printf("\n\n\n Color Map Segment created....\n");

/* obtain the RGB values of the colors into the appropriate structures */

       colors = (unsigned long *)xv_get(normal_cms, CMS_INDEX_TABLE);

       xv_get(normal_cms, CMS_X_COLORS, colors_x);
        xv_get(normal_cms, CMS_COLORS, rgb_values);

/*
        print the values to see if they are right !
        for(i=0;i<13;i++)
          printf("\n %hu %hu %hu", rgb_values[i].red,
                                rgb_values[i].green,
                                rgb_values[i].blue);

        for(i=0;i<13;i++)
          printf("\n %hu %hu %hu", colors_x[i].red,
                                colors_x[i].green,
                                colors_x[i].blue);
```

```
        printf("\n");
*/

/* Create the base frame */

        frame = (Frame)xv_create(NULL,FRAME,FRAME_LABEL,"Phoneme Maps",
        FRAME_NO_CONFIRM,FALSE,
        XV_X, 10,
        XV_Y, 10,
        XV_WIDTH,780,
        XV_HEIGHT,770,
        FRAME_INHERIT_COLORS, TRUE,
        FRAME_SHOW_HEADER,TRUE,
        FRAME_SHOW_FOOTER,TRUE,
        FRAME_LEFT_FOOTER,data_filename,
        FRAME_RIGHT_FOOTER,"Pass Number 0",NULL);


        printf("\n Frame created....\n");

/* Create the panel for the frame */


        panel = (Panel)xv_create(frame, PANEL,
                        PANEL_LAYOUT, PANEL_HORIZONTAL,
                        WIN_CMS,  control_cms,
                        NULL);


        printf("\n Panel created....\n");


/* Create the panel choices for all the vowels */

       panel_vowels = xv_create(panel, PANEL_CHOICE,
                         PANEL_LABEL_STRING, "Vowels",
                         PANEL_LABEL_BOLD, TRUE,
                         PANEL_CHOOSE_ONE, TRUE,
                         PANEL_CHOICE_STRINGS,
                            "IY",
                            "IH",
                            "EH",
                            "AE",
                            "ER",
                            "AH",
                            "AA",
                            "AO",
                            "UH",
                            "UW",
                            "NO_ACTIVATION",
                            NULL,
                         PANEL_NOTIFY_PROC, vowel_pressed,
                         NULL);
```

```
/* Create the panel choices for all the foreground and background */

      panel_fg_bg    = xv_create(panel, PANEL_CHOICE,
                         PANEL_LABEL_STRING, "Fg/Bg",
                         PANEL_LABEL_BOLD, TRUE,
                         PANEL_CHOICE_STRINGS,
                             "Foreground",
                             "Background",
                           NULL,
                         PANEL_NOTIFY_PROC, fg_bg_pressed,
                         NULL);

/* Create the panel choices for all the colors */

      panel_palette = xv_create(panel, PANEL_CHOICE,
                          PANEL_LABEL_STRING, "Colors",
                          PANEL_LABEL_BOLD, TRUE,
                          PANEL_CHOOSE_ONE, TRUE,
                            PANEL_CHOICE_XS,
            60,93,124,155,186,217,248,281,315,348,400,NULL,
                            PANEL_CHOICE_YS,
            35,NULL,
                             PANEL_NEXT_ROW, 15,
                          XV_X, (int)xv_get(panel_vowels, XV_X),
                          PANEL_CHOICE_IMAGES,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                              choice_image,
                               NULL,
                     PANEL_CHOICE_COLOR, 0, CMS_CONTROL_COLORS + MAGENTA,
                     PANEL_CHOICE_COLOR, 1, CMS_CONTROL_COLORS + RED,
                     PANEL_CHOICE_COLOR, 2, CMS_CONTROL_COLORS + ORANGE,
                     PANEL_CHOICE_COLOR, 3, CMS_CONTROL_COLORS + SALMON,
                     PANEL_CHOICE_COLOR, 4, CMS_CONTROL_COLORS + YELLOW,
                     PANEL_CHOICE_COLOR, 5, CMS_CONTROL_COLORS + SPRINGGREE
                     PANEL_CHOICE_COLOR, 6, CMS_CONTROL_COLORS + CYAN,
                     PANEL_CHOICE_COLOR, 7, CMS_CONTROL_COLORS + CORNFLOWE
                     PANEL_CHOICE_COLOR, 8, CMS_CONTROL_COLORS + BLUE,
                     PANEL_CHOICE_COLOR, 9, CMS_CONTROL_COLORS + MAROON,
                     PANEL_CHOICE_COLOR, 10, CMS_CONTROL_COLORS + WHITE,
                     PANEL_NOTIFY_PROC, vowel_color_pressed,
                     NULL);

/* Create the panel choices for the foreground  and background colors */
```

```
    panel_pal_fg_bg = xv_create(panel, PANEL_CHOICE,
                        PANEL_LABEL_STRING, "Fg/Bg Colors",
                        PANEL_LABEL_BOLD, TRUE,
                            PANEL_CHOICE_XS,
                            607, 660, NULL,
                            PANEL_CHOICE_YS,
                            35,NULL,
                        XV_X, (int)xv_get(panel_fg_bg, XV_X),
                        PANEL_CHOICE_IMAGES,
                            choice_image,
                            choice_image,
                                NULL,
                    PANEL_CHOICE_COLOR, 0, CMS_CONTROL_COLORS + GRAY,
                    PANEL_CHOICE_COLOR, 1, CMS_CONTROL_COLORS + BLACK,
                    PANEL_NOTIFY_PROC, fg_bg_color_pressed,
                    NULL);

/* Create the choices for all the maps drawn on the screen */

    panel_toggle    = xv_create(panel, PANEL_CHOICE,
                        PANEL_LABEL_STRING, "View",
                        PANEL_LABEL_BOLD, TRUE,
                            XV_X, (int)xv_get(panel_vowels, XV_X),
                            PANEL_NEXT_ROW, 15,
                        PANEL_CHOICE_STRINGS,
                            "Node Activations",
                            "Node Wins" ,
                            "Vowel Wins",
                            "Blend",
                            "Summary",
                         NULL,
                        PANEL_NOTIFY_PROC, toggle_proc ,
                        NULL);

    xv_set(panel_vowels, PANEL_CHOOSE_NONE, TRUE , NULL);
    xv_set(panel_palette, PANEL_CHOOSE_NONE, TRUE , NULL);
    xv_set(panel_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);
    xv_set(panel_pal_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);

/* Create the other buttons on the panel */

    xv_create(panel, PANEL_BUTTON,
            PANEL_LABEL_STRING, "Next Pass",
            PANEL_NEXT_ROW, 15,
            XV_X, (int) xv_get(panel_vowels, XV_X),
            PANEL_ITEM_COLOR, CMS_CONTROL_COLORS + BLACK,
            PANEL_NOTIFY_PROC, next_proc,
            NULL);

    xv_create(panel, PANEL_BUTTON,
            PANEL_LABEL_STRING, "QUIT",
            XV_X, (int) xv_get(panel_fg_bg, XV_X),
```

```
                PANEL_ITEM_COLOR, CMS_CONTROL_COLORS + BLACK,
                PANEL_NOTIFY_PROC, quit,
                NULL);

        printf("\n Buttons created....\n");

        (void)window_fit_height(panel);

/* Create the canvas on to which every thing will be drawn */

        canvas = (Canvas)xv_create(frame,CANVAS,
        CANVAS_REPAINT_PROC, canvas_repaint_proc,
        CANVAS_X_PAINT_WINDOW, TRUE,
        CANVAS_AUTO_SHRINK, FALSE,
        CANVAS_AUTO_EXPAND, FALSE,
        WIN_DYNAMIC_VISUAL, TRUE,
        WIN_CMS,          normal_cms,
        CANVAS_HEIGHT, 580,
        CANVAS_WIDTH, 680,
        CANVAS_RETAINED, FALSE,
        NULL);


        printf("\n Canvas created....\n");

        display = (Display *) xv_get(frame, XV_DISPLAY);
        planes = DefaultDepth ( display, DefaultScreen (display) );
        xid = (XID)xv_get(canvas_paint_window(canvas),XV_XID);
        can_xwin = (Window)xv_get(canvas_paint_window(canvas), XV_XID);
        screen = (Xv_screen) xv_get(frame, XV_SCREEN);


        if(!(font = XLoadQueryFont(display, "fixed") ))
        {
         fprintf(stderr,"cannot load fiixed font");
         exit(1);
        }

/* Create and initialize GC */

   gc_values.font = font->fid;
   gc_values.stipple =
        XCreateBitmapFromData( display, xid, stipple_bits, 16,2);

   gc_values.graphics_exposures = False;
   gc_values.background = colors[WHITE];

   gc =  XCreateGC(display, RootWindow(display, DefaultScreen(display)),
                GCFont | GCStipple | GCBackground | GCGraphicsExposures,
                &gc_values);

   scrollbar_v = (Scrollbar)xv_create(canvas, SCROLLBAR,
                        SCROLLBAR_DIRECTION, SCROLLBAR_VERTICAL,
```

```
                    SCROLLBAR_SPLITTABLE, TRUE,
                    NULL);

    scrollbar_h = (Scrollbar)xv_create(canvas, SCROLLBAR,
                    SCROLLBAR_DIRECTION, SCROLLBAR_HORIZONTAL,
                    SCROLLBAR_SPLITTABLE, TRUE,
                    NULL);


    cmap = DefaultColormap(display, DefaultScreen(display) );

    printf("\n GC created  and CMap obtained....\n");

/* Open and Read the first set of data from the data file */

    datafile_fd = fopen("data","r");

    fscanf(datafile_fd,"%d", &pass_num);
    fscanf(datafile_fd,"%d", &output_x);
    fscanf(datafile_fd,"%d", &output_y);

    for(row =0; row < output_x; row ++)
     {
         for(col =0; col <output_y; col ++)
          {
           max_table[row][col] = 0;
           for(vowel=0;vowel <10; vowel++)
            {
                fscanf(datafile_fd,"%d", &table[vowel][row][col]);
                max_table[row][col] += table[vowel][row][col];
            }
          }
        }

    printf("\n File opened\n");

/* Create the Pixmaps */

        nodes_map = XCreatePixmap(display,
                (XID) xv_get(canvas_paint_window(canvas),XV_XID),
                770, 770, planes);
        printf("\n Pixmap 1 Creation succeeded\n");

        win_map = XCreatePixmap(display,
                (XID) xv_get(canvas_paint_window(canvas),XV_XID),
                770, 770, planes);
        printf("\n Pixmap 2 Creation succeeded\n");

        summary_map = XCreatePixmap(display,
                (XID) xv_get(canvas_paint_window(canvas),XV_XID),
                770, 770, planes);
        printf("\n Pixmap 3 Creation succeeded\n");
```

```
        blend_map = XCreatePixmap(display,
                (XID) xv_get(canvas_paint_window(canvas),XV_XID),
                770, 670, planes);
        printf("\n Pixmap 4 Creation succeeded\n");

        dummy_map = XCreatePixmap(display,
                (XID) xv_get(canvas_paint_window(canvas),XV_XID),
                770, 670, planes);


        v_icon = (Icon) xv_create(frame, ICON,
                ICON_IMAGE, vowel_image,
                XV_X,10,
                XV_Y,0,
                NULL);

        xv_set(frame, FRAME_ICON, v_icon, NULL);

        printf("\n Icon created and Set \n");

/* Enter the main loop for reading events */

       xv_main_loop(frame);

}

/*
This is the procedure that is determines which choice was selected and
which  map will be drawn on to the screen
*/

void
toggle_proc(item, event)
Panel_item item;
Event *event;
{
    int choice;

         color_display_flag = -1;
         choice = (int) xv_get(item, PANEL_VALUE);
         if(choice == 0)
          {
          toggle_flag = FALSE;
          compress_flag = FALSE;
          new_toggle_flag = FALSE;
          }
         else if(choice ==1)
          {
          toggle_flag = TRUE;
          compress_flag = FALSE;
          new_toggle_flag = FALSE;
          }
         else if(choice ==2)
```

```
        {
       new_toggle_flag = TRUE;
        }
       else if(choice ==3)
        {
        toggle_flag = FALSE;
        compress_flag = TRUE;
        new_toggle_flag = FALSE;
        }
       else if(choice ==4)
        {
        toggle_flag = TRUE;
        compress_flag = TRUE;
        new_toggle_flag = FALSE;
        }


/* can_xvwin = (Xv_Window)xv_get(canvas_paint_window(canvas), XV_XID);*/

        xv_set(panel_palette, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_vowels, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_pal_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);

/* Clean up the canvas and redraw */

        XClearWindow(display, can_xwin);
        canvas_repaint_proc(canvas,
                (Xv_Window) xv_get(canvas_paint_window(canvas), XV_XID) ,
                display, can_xwin, (Xv_xrectlist *) NULL);

}

/*
This is the call back procedure that displays the notices when the choices
for the vowels are selected
*/


Xv_opaque
vowel_pressed(item, value, event)
Panel_item item;
int value;
Event *event;
{
        int choice;
        int answer;
        char vowel_str[3];
        Panel panel;
        Frame frame;

        choice = (int) xv_get(item, PANEL_VALUE);
```

```
        panel = xv_get(item, PANEL_PARENT_PANEL);
        frame = xv_get ( xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);

        sprintf(vowel_str,"%d",choice);

        if(choice >=0 && choice <10)
        {
                answer = notice_prompt(  panel, NULL,
                NOTICE_MESSAGE_STRINGS,
                "This vowel is vowel number", vowel_str, " in the data file",
                "The vowel has the arpabet symbol ",vowel_array[choice],
                "An example of this vowel occurring in a word would be ",
                example_array[choice],
                "The color associated with this vowel is ", color_array[choice],
                "There is a box that appears below ", vowel_array[choice],
                " for this color", NULL,
                NOTICE_BUTTON_YES, "OK",
                NOTICE_NO_BEEPING, TRUE,
                NULL);
        }

        else if(choice == 10)
        {
                answer = notice_prompt(  panel, NULL,
                NOTICE_MESSAGE_STRINGS,
                "This represents a node that has not been activated",
                "in this pass",
                "The color associated with this node is ",
                "WHITE",
                "There is a box that appears below this choice",
                " for this color", NULL,
                NOTICE_BUTTON_YES, "OK",
                NOTICE_NO_BEEPING, TRUE,
                NULL);
        }

}

/*
This is the call back procedure that displays the notice when the choices
for the vowel colors are selected. The canvas is redrawn and all locations
where this vowel appears in displayed
*/

Xv_opaque
vowel_color_pressed(item, value,  event)
Panel_item item;
int value;
Event *event;
{
        int choice;
        int answer;
        char color_str[3];
```

```
        Panel panel;
        Frame frame;

        color_display_flag = -1;
        choice = (int) xv_get(item, PANEL_VALUE);

        panel = xv_get(item, PANEL_PARENT_PANEL);

        frame = xv_get ( xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);

        if(choice >=0 && choice <10)
        {
        answer = notice_prompt(  panel, NULL,
        NOTICE_MESSAGE_STRINGS,
        "This color is  ", color_array[choice],
        "and represents the vowel ", vowel_array[choice],
        "The corresponding node appears ABOVE",
        "You will now see all nodes where ", color_array[choice],
        "appears in the grid", "The other regions where activations ",
        "were present",
        "will be painted with the background color", NULL,
        NOTICE_BUTTON_YES, "OK",
        NOTICE_NO_BEEPING, TRUE,
        NULL);


         }

        else if(choice == 10)
        {
        answer = notice_prompt(  panel, NULL,
        NOTICE_MESSAGE_STRINGS,
        "This color is  ", color_array[choice],
        " and represents a node that has not been activated ",
        "The corresponding node appears ABOVE",
        "You will now see all nodes where ", color_array[choice],
        "appears in the grid", "The other regions where activations ",
        "were present",
        "will be painted with the background color", NULL,
                NOTICE_BUTTON_YES, "OK",
                NOTICE_NO_BEEPING, TRUE,
                NULL);
        }
          color_display_flag = choice;
          XClearWindow(display, can_xwin);
          canvas_repaint_proc(canvas,
                (Xv_Window) xv_get(canvas_paint_window(canvas), XV_XID) ,
                display, can_xwin, (Xv_xrectlist *) NULL);
}

/*
This is the call back procedure that displays the notice when the choices
for the foreground background are selected.
```

```
*/

Xv_opaque
fg_bg_pressed(item, value,  event)
Panel_item item;
int value;
Event *event;
{
        int choice;
        int answer;
        Panel panel;
        Frame frame;

        panel = xv_get(item, PANEL_PARENT_PANEL);
        frame = xv_get ( xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
        choice = (int) xv_get(item, PANEL_VALUE);

                answer = notice_prompt(   panel, NULL,
                NOTICE_MESSAGE_STRINGS,
                "This choice represents the ", fg_bg_array[choice],
                "The color associated with it is ",
                fg_bg_color_array[choice],
                "There is a box that appears below this choice  that",
                "shows this color",
                NULL,
                NOTICE_BUTTON_YES, "OK",
                NOTICE_NO_BEEPING, TRUE,
                NULL);


}

/*
This is the call back procedure that displays the notice when the choices
for the foreground  and background colors are selected.
*/

Xv_opaque
fg_bg_color_pressed(item, value,  event)
Panel_item item;
int value;
Event *event;
{
        int choice;
        int answer;
        Panel panel;
        Frame frame;

        panel = xv_get(item, PANEL_PARENT_PANEL);
        frame = xv_get ( xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
        choice = (int) xv_get(item, PANEL_VALUE);

                answer = notice_prompt(   panel, NULL,
```

```
                NOTICE_MESSAGE_STRINGS,
                "This color is  ", fg_bg_color_array[choice],
                "and represents the ", fg_bg_array[choice],
                NULL,
                NOTICE_BUTTON_YES, "OK",
                NOTICE_NO_BEEPING, TRUE,
                NULL);


}

/*
This is the call proicedure that is activated when the next pass button
is selected. A new set of data is read from the data file and the maps
are redrawn. If the end of file is reached the data file is rewound.
*/

void
next_proc(item,event)
Panel_item item;
Event *event;
{
        int row, col, vowel;
        int answer=0;

        Panel local_panel;

        if( (fscanf(datafile_fd,"%d", &pass_num)) == EOF)
        {
                answer = notice_prompt(  xv_get(item, PANEL_PARENT_PANEL), NULL,
                NOTICE_MESSAGE_STRINGS,
                " End of Data file reached, Rewinding it ..!",
                NULL,
                NOTICE_BUTTON_YES, "YES",
                NULL);

          rewind(datafile_fd);
          fscanf(datafile_fd,"%d", &pass_num);
         }
        for(row =0; row < output_x; row ++)
         {
          for(col =0; col <output_y; col ++)
           {
            max_table[row][col] = 0;
            for(vowel=0;vowel <10; vowel++)
             {
                fscanf(datafile_fd,"%d", &table[vowel][row][col]);
                max_table[row][col] += table[vowel][row][col];
             }
           }
         }

        draw_in_pixmap();
```

```
}

/* This is the function that is called for drawing the pixmaps */

draw_in_pixmap()
{
        int width, height;
        char num_x[5],num_y[5],tmp[5], num_s[5];
        int x1,x2 =0;
        int y1,y2 =0;
        int flag=False;
        int c =0;
        int i =0;
        int j =0;
        int present_in_array = FALSE;
        int check = 0;
        int max_x=0;
        int max_y =0;
        int max_num[10][2];
        int max_times = 0;
        int max_vowel = -1;
        int x_pos = 90;
        int y_pos = 50;
        int str_x = 75;
        int str_y = 40;
        int percent=0;
        int start_pos_x, start_pos_y;
        int row, col, vowel, n;
        int number_rows, number_pixels,  extra_pixels,pix_count;
        int x, y, count, sum ;
        int v_arr[10], index;
        unsigned long int p_red =0;
        unsigned long int p_blue =0;
        unsigned long int p_green =0;
        double percentage;
        unsigned long int pixel_value= 0;
        char pass_char[20];
        int answer;
        int b_sum =0;

 /* Clean up existing  pixmaps */

        XCopyArea(display, dummy_map, nodes_map, gc, 0,0, 770, 770, 0,0);
        XCopyArea(display, dummy_map, nodes_map, gc, 0,0, 770, 770, 0,0);
        XCopyArea(display, dummy_map, win_map, gc, 0,0, 770, 770, 0,0);
        XCopyArea(display, dummy_map, summary_map, gc, 0,0, 770, 770, 0,0);
        XCopyArea(display, dummy_map, blend_map, gc, 0,0, 770, 770, 0,0);

 /* Start drawing the blend map */
 /* This creates a new color based on the vowels that activate a node */
 /* The algorithm has been presented in the earlier sections */

    for(x=0; x < Output_X; x++ )
```

```
{
for(y=0;y<Output_Y;y++ )
  {

   for(i=0; i< 10; i++)
        v_arr[i]=0;

     for(vowel=0,count=0,b_sum=0 ;vowel <10; vowel++)
        {
         if(table[vowel][x][y] >0)
          {
            b_sum = b_sum + table[vowel][x][y];
            v_arr[count++]=  vowel;
          }

         }

        pixel.red = 0;
        pixel.blue = 0;
        pixel.green = 0;

     if(count!=0)
        {

         for(index = 0; index < count; index++)
         {

          percentage = 100.0 *(double) table[v_arr[index]][x][y]/(double)b_sum;
           percent = (int) percentage;
           pixel.red +=  colors_x[v_arr[index]+1 ].red * percent /100;
           pixel.green +=  colors_x[v_arr[index]+1 ].green * percent /100;
           pixel.blue +=  colors_x[v_arr[index]+1 ].blue * percent /100;
          }

         }
        pixel.flags = DoRed|DoGreen|DoBlue;

XAllocColor(display, cmap, &pixel );


XSetForeground(display, gc, colors[BLACK]);
strcpy(num_x,"");
strcat(num_x," ");
sprintf(tmp,"%d",x);
strcat(num_x,tmp);
strcat(num_x," ");
XDrawString(display, blend_map, gc,str_x +30 + x *62 ,str_y , num_x, strlen(num_x) );

strcpy(num_y,"");
strcat(num_y," ");
sprintf(tmp,"%d",y);
strcat(num_y,tmp);
strcat(num_y," ");
```

```
        XSetForeground(display, gc, colors[BLACK]);
        XDrawString(display, blend_map, gc,str_x ,str_y +40 +y *62 , num_y, strlen(num_y) );

         start_pos_x = x_pos + x * 62;
         start_pos_y = y_pos + y * 62;


            if( b_sum >0 )
            {
            XSetForeground(display,gc, pixel.pixel );

            XFillRectangle(display, blend_map, gc,  start_pos_x ,start_pos_y , 60 , 60);
               }

            else
               {
                XSetForeground(display,gc, colors[WHITE] );
               XFillRectangle(display, blend_map, gc,  start_pos_x ,start_pos_y , 60 , 60);
               }


          }

        }



/* Start drawing the nodes map */
/* This shows all activations for all nodes */
/* The algorithm has been presented in the earlier sections */

        for(x=0;x <output_x; x++)
        {
          XSetForeground(display, gc, colors[BLACK]);
          strcpy(num_x,"");
          strcat(num_x," ");
          sprintf(tmp,"%d",x);
          strcat(num_x,tmp);
          strcat(num_x," ");
          XDrawString(display, nodes_map, gc,str_x +30 + x *62 ,str_y , num_x, strlen(num_x) );

         for(y= 0; y < output_y; y++)
         {
           strcpy(num_y,"");
           strcat(num_y," ");
           sprintf(tmp,"%d",y);
           strcat(num_y,tmp);
           strcat(num_y," ");
           XSetForeground(display, gc, colors[BLACK]);
           XDrawString(display, nodes_map, gc,str_x ,str_y +40 +y *62 , num_y, strlen(num_y) )

           start_pos_x = x_pos + x * 62;
           start_pos_y = y_pos + y * 62;
```

```
for(i=0; i< 10; i++)
   v_arr[i]=0;


for(vowel=0,count=0,sum=0 ;vowel <10; vowel++)
  {
   if(table[vowel][x][y] >0)
    {
     sum = sum + table[vowel][x][y];
     v_arr[count++]=  vowel;
    }

 }

 if(count!=0)
   {

for(index = 0, pix_count=0; index < count; index++)
    {

     percentage =(double) table[v_arr[index]][x][y]/(double)sum;

     number_pixels = nint (60.0 * 60.0 *  percentage);

     number_rows = number_pixels / 60;
     extra_pixels = number_pixels % 60;



     for(n = 1; n <= number_pixels; n++)
       {

         if( pix_count  ==  60)
          {
           start_pos_x = start_pos_x -60 ;
           start_pos_y = start_pos_y + 1 ;
           pix_count =0;
          }

        XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );
        XDrawPoint(display, nodes_map, gc, start_pos_x, start_pos_y);
        pix_count++;
        start_pos_x++;
       }
     }

   }

 else
    {
     XSetForeground(display,gc, colors[WHITE] );
    XFillRectangle(display, nodes_map, gc,  start_pos_x ,start_pos_y , 60 , 60);
```

```
            }

          }

        }

/* Start drawing the winninmg nodes map */
/* This shows the vowels for which each node responded the best */

        for(x =0; x < output_x; x++)
        {
         XSetForeground(display, gc, colors[BLACK]);
         strcpy(num_x,"");
         strcat(num_x," ");
         sprintf(tmp,"%d",x);
         strcat(num_x,tmp);
         strcat(num_x," ");
         XDrawString(display, win_map, gc,str_x +30 + x *62 ,str_y , num_x, 3 );
         for(y =0; y <output_y; y++)
          {


           strcpy(num_y,"");
           strcat(num_y," ");
           sprintf(tmp,"%d",y);
           strcat(num_y,tmp);
           strcat(num_y," ");
           XSetForeground(display, gc, colors[BLACK]);
           XDrawString(display, win_map, gc,str_x ,str_y +40 +y *62 , num_y, strlen(num_y) );

           start_pos_x = x_pos + x * 62;
           start_pos_y = y_pos + y * 62;
           max_times=0;
           max_vowel = -1;

            for(i=0; i< 10; i++)
              v_arr[i]=0;

           count =0;

           for(vowel=0 ;vowel <10; vowel++)
          {
              if(table[vowel][x][y] >max_times)
              {
                      count =0;
                      max_times = table[vowel][x][y];
                      max_vowel = vowel;
                      v_arr[count] = vowel;
              }
              else if(table[vowel][x][y] == max_times && max_times !=0)
              {
```

```
                v_arr[++count] = vowel;
            }
        }


        if (count >0 )
            {
        for(index = 0, pix_count=0; index <= count; index++)
            {

            percentage = 1.0 /(double)(count +1);

            number_pixels = nint (60.0 * 60.0 *  percentage);

            number_rows = number_pixels / 60;
            extra_pixels = number_pixels % 60;

            for(n = 1; n <= number_pixels; n++)
              {

                if( pix_count  ==  60)
                 {
                  start_pos_x = start_pos_x - 60 ;
                  start_pos_y = start_pos_y + 1 ;
                  pix_count =0;
                 }

               XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );
               XDrawPoint(display, win_map, gc, start_pos_x, start_pos_y);
               pix_count++;
               start_pos_x++;
              }
            }
            }

        else if( max_times == 0)
            {
            XSetForeground(display, gc, colors[WHITE]);
             XFillRectangle(display, win_map, gc,  x_pos + x* 62 ,y_pos + 62  *y, 60 , 60);
            }
        else
            {

             XSetForeground(display, gc, colors[max_vowel+1]);
             XFillRectangle(display, win_map, gc,  x_pos + x* 62 ,y_pos + 62  *y, 60 , 60);
            }

    }

}

 printf("\n Gets Boring.....huh ??    :-(  \n");
```

```
/* Start drawing the summary map */
/* This shows the  summary information of all maps */


        for(x=0;x <output_x; x++)
        {
          XSetForeground(display, gc, colors[BLACK]);
          strcpy(num_x,"");
          strcat(num_x," ");
          sprintf(tmp,"%d",x);
          strcat(num_x,tmp);
          strcat(num_x," ");
          XDrawString(display, summary_map, gc,str_x +20 + x *31 ,str_y , num_x, strlen(num_

         for(y= 0; y < output_y; y++)
         {
           strcpy(num_y,"");
           strcat(num_y," ");
           sprintf(tmp,"%d",y);
           strcat(num_y,tmp);
           strcat(num_y," ");
           XSetForeground(display, gc, colors[BLACK]);
           XDrawString(display, summary_map, gc,str_x ,str_y +25 +y *31 , num_y, strlen(num_

           start_pos_x = x_pos + x * 31;
           start_pos_y = y_pos + y * 31;

            for(i=0; i< 10; i++)
               v_arr[i]=0;

            for(vowel=0,count=0,sum=0 ;vowel <10; vowel++)
              {
               if(table[vowel][x][y] >0)
                {
                  sum = sum + table[vowel][x][y];
                  v_arr[count++]=  vowel;
                }

             }


           if(count!=0)
              {

             for(index = 0, pix_count=0; index < count; index++)
              {

                percentage =(double) table[v_arr[index]][x][y]/(double)sum;

                number_pixels = nint (30.0 * 30.0 *  percentage);

                number_rows = number_pixels / 30;
                extra_pixels = number_pixels % 30;
```

```
        for(n = 1; n <= number_pixels; n++)
          {

            if( pix_count == 30)
             {
              start_pos_x = start_pos_x - 30 ;
              start_pos_y = start_pos_y + 1 ;
              pix_count =0;
             }

           XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );
           XDrawPoint(display, summary_map, gc, start_pos_x, start_pos_y);
           pix_count++;
           start_pos_x++;
          }
        }

     }

    else
        {
         XSetForeground(display,gc, colors[WHITE] );
        XFillRectangle(display, summary_map, gc,  start_pos_x ,start_pos_y , 30 , 30);
       }


   }


}

str_x = 385 ;
x_pos = 400;

for(x =0; x < output_x; x++)
{
 XSetForeground(display, gc, colors[BLACK]);
 strcpy(num_x,"");
 strcat(num_x," ");
 sprintf(tmp,"%d",x);
 strcat(num_x,tmp);
 strcat(num_x," ");
 XDrawString(display, summary_map, gc,str_x +25 + x *31 ,str_y , num_x, strlen(num_x

 for(y =0; y <output_y; y++)
  {
   start_pos_x = x_pos + x * 31;
   start_pos_y = y_pos + y * 31;

   strcpy(num_y,"");
```

```
    strcat(num_y," ");
    sprintf(tmp,"%d",y);
    strcat(num_y,tmp);
    strcat(num_y," ");
    XSetForeground(display, gc, colors[BLACK]);
    XDrawString(display, summary_map, gc,str_x ,str_y +25 +y *31 , num_y, strlen(num_

    max_times=0;
    max_vowel = -1;

     for(i=0; i< 10; i++)
        v_arr[i]=0;

    count =0;

    for(vowel=0 ;vowel <10; vowel++)
   {
        if(table[vowel][x][y] >max_times)
        {
                count =0;
                max_times = table[vowel][x][y];
                max_vowel = vowel;
                v_arr[count] = vowel;
        }
        else if(table[vowel][x][y] == max_times && max_times !=0)
        {
                v_arr[++count] = vowel;
        }
    }


     if (count >0 )
        {
  for(index = 0, pix_count=0; index <= count; index++)
        {

         percentage = 1.0 /(double)(count +1);

         number_pixels = nint (30.0 * 30.0 *  percentage);

         number_rows = number_pixels / 30;
         extra_pixels = number_pixels % 30;

         for(n = 1; n <= number_pixels; n++)
           {

             if( pix_count  ==  30)
              {
               start_pos_x = start_pos_x - 30 ;
               start_pos_y = start_pos_y + 1 ;
               pix_count =0;
              }
```

```
                XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );
                XDrawPoint(display, summary_map, gc, start_pos_x, start_pos_y);
                pix_count++;
                start_pos_x++;
              }
            }
            }

          else if( max_times == 0)
              {
              XSetForeground(display, gc, colors[WHITE]);
               XFillRectangle(display, summary_map, gc,  x_pos + x* 31 ,y_pos + 31  *y, 30 , 3
              }
          else
              {

               XSetForeground(display, gc, colors[max_vowel+1]);
               XFillRectangle(display, summary_map, gc,  x_pos + x* 31 ,y_pos + 31  *y, 30 , 3

              }

          strcpy(num_s,"");
          sprintf(tmp,"%d",max_table[x][y]);
          strcat(num_s,tmp);
          strcat(num_s,"");
          XSetForeground(display, gc, colors[BLACK]);
          XDrawString(display, summary_map, gc, x_pos + x * 31 + 10,
                            y_pos + y* 31 + 18, num_s, strlen(num_s) );

        }

      }
/********************************** Vowel Wins ***************************/

       x_pos = 90;
       y_pos = 330;
       str_x = 75;
       str_y = 320;

       for( vowel = 0; vowel <10; vowel ++)
      {
        max_times = -1;
        max_x = -1;
       max_y = -1;
        for(x=0; x<output_x; x++)
       {
          for(y=0;y<output_y;y++)
           {

              if(table[vowel][x][y] >max_times)
               {
                max_times = table[vowel][x][y];
                max_x = x;
```

```
            max_y = y;
           }

       }
     }
    max_num[vowel][0] = max_x;
    max_num[vowel][1] = max_y;
}

for(x=0;x <output_x; x++)
{
   XSetForeground(display, gc, colors[BLACK]);
   strcpy(num_x,"");
   strcat(num_x," ");
   sprintf(tmp,"%d",x);
   strcat(num_x,tmp);
   strcat(num_x," ");
   XDrawString(display, summary_map, gc,str_x +25 + x *31 ,str_y , num_x, strlen(num_

  for(y= 0; y < output_y; y++)
  {
    strcpy(num_y,"");
    strcat(num_y," ");
    sprintf(tmp,"%d",y);
    strcat(num_y,tmp);
    strcat(num_y," ");
    XSetForeground(display, gc, colors[BLACK]);
    XDrawString(display, summary_map, gc,str_x ,str_y +25 +y *31 , num_y, strlen(num_

    start_pos_x = x_pos + x * 31;
    start_pos_y = y_pos + y * 31;

     for(i=0; i< 10; i++)
        v_arr[i]=0;


     for(vowel=0,count=0,sum=0 ;vowel <10; vowel++)
        {
        if(x == max_num[vowel][0] && y == max_num[vowel][1])
         {
           sum = sum + table[vowel][x][y];
           v_arr[count++]=  vowel;
         }

        }

     if(count> 1 )
        {
         for(index = 0, pix_count=0; index < count; index++)
         {

            percentage =(double) table[v_arr[index]][x][y]/(double)sum;
            number_pixels = nint (30.0 * 30.0 *  percentage);
```

```
                number_rows = number_pixels / 30;
                extra_pixels = number_pixels % 30;

                XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );

                for(n = 1; n <= number_pixels; n++)
                {
                 if( pix_count  ==  30)
                   {
                    start_pos_x = start_pos_x -30 ;
                    start_pos_y = start_pos_y + 1 ;
                    pix_count =0;
                   }

                 XDrawPoint(display, summary_map, gc, start_pos_x, start_pos_y);
                 pix_count++;
                 start_pos_x++;
                } /* end of n loop */

               } /* end of index loop */

             } /* end of count ne 0 */


          else if(count ==1)
             {
              XSetForeground(display,gc, colors[v_arr[0] +1 ] );
              XFillRectangle(display, summary_map, gc,  start_pos_x ,start_pos_y , 30 , 30);
             }
          else
             {
              XSetForeground(display,gc, colors[BLACK] );
              XFillRectangle(display, summary_map, gc,  start_pos_x ,start_pos_y , 30 , 30);
             }


        }   /* y loop */


     }   /* x loop */

/********************************** Vowel Wins Ends ***************************/

/********************************** Blend Map ***************************/
     x_pos = 400;
     y_pos = 330;
     str_y = 320;
     str_x = 385;

  for(x=0; x < output_x; x++ )
     {

      for(y=0;y<output_y;y++ )
```

```
{
 for(i=0; i< 10; i++)
      v_arr[i]=0;

   for(vowel=0,count=0,b_sum=0 ;vowel <10; vowel++)
      {
       if(table[vowel][x][y] >0)
        {
          b_sum = b_sum + table[vowel][x][y];
          v_arr[count++]=  vowel;
        }

       }

      pixel.red = 0;
      pixel.blue = 0;
      pixel.green = 0;

   if(count!=0)
      {

        for(index = 0; index < count; index++)
        {

         percentage = 100.0 *(double) table[v_arr[index]][x][y]/(double)b_sum;
          percent = (int) percentage;
          pixel.red +=  colors_x[v_arr[index]+1 ].red * percent /100;
          pixel.green +=  colors_x[v_arr[index]+1 ].green * percent /100;
          pixel.blue +=  colors_x[v_arr[index]+1 ].blue * percent /100;
        }

       }
      pixel.flags = DoRed|DoGreen|DoBlue;

XAllocColor(display, cmap, &pixel );


XSetForeground(display, gc, colors[BLACK]);
strcpy(num_x,"");
strcat(num_x," ");
sprintf(tmp,"%d",x);
strcat(num_x,tmp);
strcat(num_x," ");
XDrawString(display, summary_map, gc,str_x +25 + x *31 ,str_y , num_x, strlen(num_x)

strcpy(num_y,"");
strcat(num_y," ");
sprintf(tmp,"%d",y);
strcat(num_y,tmp);
strcat(num_y," ");
XSetForeground(display, gc, colors[BLACK]);
XDrawString(display, summary_map, gc,str_x ,str_y +25 +y *31 , num_y, strlen(num_y) )
```

```
        start_pos_x = x_pos + x * 31;
        start_pos_y = y_pos + y * 31;


            if( b_sum >0 )
            {
            XSetForeground(display,gc, pixel.pixel );

            XFillRectangle(display, summary_map, gc,  start_pos_x ,start_pos_y , 30 , 30);
              }

            else
               {
                XSetForeground(display,gc, colors[WHITE] );
               XFillRectangle(display, summary_map, gc,  start_pos_x ,start_pos_y , 30 , 30);
              }


         }

    }
/**********************************Blend Ends ***************************/


        /* end of 4 pixmaps creation */

        printf("\n Pixmaps (re)drawn \n");

        compress_flag = FALSE ;
        toggle_flag =FALSE;

/* Addition for RTG */

        xv_set(panel_palette, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_vowels, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_pal_fg_bg, PANEL_CHOOSE_NONE, TRUE , NULL);
        xv_set(panel_toggle, PANEL_VALUE, 0 , NULL);

        XClearWindow(display, can_xwin);
        canvas_repaint_proc(canvas,
                (Xv_Window) xv_get(canvas_paint_window(canvas), XV_XID) ,
                display, can_xwin, (Xv_xrectlist *) NULL);



}

/*
This is the call back procedure thst is called when the quit button is selected
*/
```

```
void
quit(item,event)
Panel_item item;
Event *event;
{
        int answer;

        Panel panel = xv_get(item, PANEL_PARENT_PANEL);
        Frame frame = xv_get ( xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);

                answer = notice_prompt(    panel, NULL,
                NOTICE_FOCUS_XY, event_x(event), event_y(event),
                NOTICE_MESSAGE_STRINGS,
                "Are you sure you want to quit ??",
                NULL,
                NOTICE_BUTTON_YES, "YES",
                NOTICE_BUTTON_NO, "NO",
                NULL);

  if(answer == NOTICE_YES)
   {
   exit(0);
   }
}

/*
This is the call back procedure that for repainting the canvas.
*/

void
canvas_repaint_proc(canvas, paint_window, dpy, xwin, xrects)
Canvas canvas;
Xv_Window paint_window;
Display *dpy;
Window xwin;
Xv_xrectlist *xrects;
{
        int width, height;
        char num_x[5],num_y[5],tmp[5], num_s[5];
        int x1,x2 =0;
        int y1,y2 =0;
        int flag=False;
        int c =0;
        int i =0;
        int j =0;
        int present_in_array = FALSE;
        int check = 0;
        int max_times = 0;
        int max_x=0;
        int max_y =0;
        int max_num[10][2];
        int max_vowel = -1;
        int x_pos = 90;
```

```
        int y_pos = 50;
        int str_x = 75;
        int str_y = 40;
        int percent=0;
        int start_pos_x, start_pos_y;
        int row, col, vowel, n;
        int number_rows, number_pixels,  extra_pixels,pix_count;
        int x, y, count, sum ;
        int v_arr[10], index;
        double percentage;
        unsigned long int pixel_value= 0;
        char pass_char[20];
        int answer;
        int b_sum;


Frame frame =  xv_get( canvas, XV_OWNER);

/* For the first set of data */

if(just_created == TRUE )
  {
   XClearWindow(display, can_xwin);
   XCopyArea(display, can_xwin, dummy_map, gc, 0,0, 770, 770, 0,0);
  just_created = FALSE;
   draw_in_pixmap();
  }

/* Update the footer to show the pass number */

        strcpy(pass_char,"");
        strcat(pass_char,"Pass Number ");
        sprintf(tmp,"%d",pass_num);
        strcat(pass_char,tmp);
        xv_set(frame, FRAME_RIGHT_FOOTER,pass_char,NULL);


/* if a "Vowel colors" button has been depressed */

 if(color_display_flag > -1)
  {

        for(x=0;x <output_x; x++)
        {
          XSetForeground(display, gc, colors[BLACK]);
          strcpy(num_x,"");
          strcat(num_x," ");
          sprintf(tmp,"%d",x);
          strcat(num_x,tmp);
          strcat(num_x," ");
          XDrawString(display, can_xwin, gc,str_x +30 + x *62 ,str_y , num_x, strlen(num_x) );

         for(y= 0; y < output_y; y++)
```

```
{
  strcpy(num_y,"");
  strcat(num_y," ");
  sprintf(tmp,"%d",y);
  strcat(num_y,tmp);
  strcat(num_y," ");
  XSetForeground(display, gc, colors[BLACK]);
  XDrawString(display, can_xwin, gc,str_x ,str_y +40 +y *62 , num_y, strlen(num_y) );

  start_pos_x = x_pos + x * 62;
  start_pos_y = y_pos + y * 62;

  for(i=0; i< 10; i++)
     v_arr[i]=0;


  for(vowel=0,count=0,sum=0 ;vowel <10; vowel++)
     {
      if(table[vowel][x][y] >0)
       {
        sum = sum + table[vowel][x][y];
        v_arr[count++]=  vowel;
       }

     }

  if(count!=0  && color_display_flag != 10)
     {
      present_in_array=FALSE;
      for(check=0;check<count;check++)
       {
        if(v_arr[check] == color_display_flag)
            present_in_array = TRUE;
       }
     if(present_in_array == TRUE)
      {
      for(index = 0, pix_count=0; index < count; index++)
      {

          percentage =(double) table[v_arr[index]][x][y]/(double)sum;
          number_pixels = nint (60.0 * 60.0 *  percentage);
          number_rows = number_pixels / 60;
          extra_pixels = number_pixels % 60;

          for(n = 1; n <= number_pixels; n++)
          {
           if( pix_count  ==  60)
             {
              start_pos_x = start_pos_x -60 ;
              start_pos_y = start_pos_y + 1 ;
              pix_count =0;
             }
```

```
                    if(color_display_flag == v_arr[index] )
                     XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );
                    else
                     XSetForeground(display,gc, colors[BLACK] );
                     XDrawPoint(display, can_xwin, gc, start_pos_x, start_pos_y);
                     pix_count++;
                     start_pos_x++;
                    } /* end of n loop */
                   } /* end of index loop */

                  } /* end of present in array loop */

                  else if(present_in_array == FALSE)
                   {
                    XSetForeground(display,gc, colors[BLACK] );
                    XFillRectangle(display, can_xwin, gc, start_pos_x, start_pos_y, 60, 60);
                    }


                  } /* end of count ne 0 */
              else if(count!=0  && color_display_flag == 10)
                 {
                  XSetForeground(display,gc, colors[BLACK] );
                  XFillRectangle(display, can_xwin, gc,  start_pos_x ,start_pos_y , 60, 60);
                 }

              else
                 {
                  XSetForeground(display,gc, colors[WHITE] );
                  XFillRectangle(display, can_xwin, gc,  start_pos_x ,start_pos_y , 60 , 60);
                 }


            }   /* y loop */


        }   /* x loop */

    } /* end of color_display */


/*
if other butons have been depressed  copy the appropriate pixmap  on to the
screen
*/

 else if( new_toggle_flag == FALSE)
  {
   if(compress_flag == FALSE && toggle_flag ==FALSE)
    {
        XCopyArea(display, nodes_map, can_xwin, gc, 0,0, 770, 770, 0,0);
    }
```

```
  else if(compress_flag ==FALSE && toggle_flag ==TRUE )
   {
       XCopyArea(display, win_map, can_xwin, gc, 0,0,770,770,0,0);
   }

  else if(compress_flag == TRUE && toggle_flag ==FALSE)
   {
      XCopyArea(display, blend_map, can_xwin, gc, 0,0,770,770,0,0);

   }
  else if(compress_flag == TRUE && toggle_flag ==TRUE)
   {

       XCopyArea(display, summary_map, can_xwin, gc, 0,0,770,770,0,0);
   }
  }

/* otherwise the vowel winning information has to be displayed */
/* This tell you the ten nodes that responded best to the 10 vowels */

   else if(new_toggle_flag  == TRUE )
    {
        for( vowel = 0; vowel <10; vowel ++)
       {
         max_times = -1;
         max_x = -1;
        max_y = -1;
         for(x=0; x<output_x; x++)
        {
           for(y=0;y<output_y;y++)
            {

               if(table[vowel][x][y] >max_times)
                {
                 max_times = table[vowel][x][y];
                 max_x = x;
                 max_y = y;
                }

            }
          }
         max_num[vowel][0] = max_x;
         max_num[vowel][1] = max_y;
       }

       for(x=0;x <output_x; x++)
       {
         XSetForeground(display, gc, colors[BLACK]);
         strcpy(num_x,"");
         strcat(num_x," ");
         sprintf(tmp,"%d",x);
         strcat(num_x,tmp);
         strcat(num_x," ");
```

```
 XDrawString(display, can_xwin, gc,str_x +30 + x *62 ,str_y , num_x, strlen(num_x) );

for(y= 0; y < output_y; y++)
{
  strcpy(num_y,"");
  strcat(num_y," ");
  sprintf(tmp,"%d",y);
  strcat(num_y,tmp);
  strcat(num_y," ");
  XSetForeground(display, gc, colors[BLACK]);
  XDrawString(display, can_xwin, gc,str_x ,str_y +40 +y *62 , num_y, strlen(num_y) );

  start_pos_x = x_pos + x * 62;
  start_pos_y = y_pos + y * 62;

   for(i=0; i< 10; i++)
     v_arr[i]=0;


   for(vowel=0,count=0,sum=0 ;vowel <10; vowel++)
      {
       if(x == max_num[vowel][0] && y == max_num[vowel][1])
        {
          sum = sum + table[vowel][x][y];
          v_arr[count++]=  vowel;
        }

       }

   if(count> 1 )
      {
       for(index = 0, pix_count=0; index < count; index++)
       {

          percentage =(double) table[v_arr[index]][x][y]/(double)sum;
          number_pixels = nint (60.0 * 60.0 *  percentage);
          number_rows = number_pixels / 60;
          extra_pixels = number_pixels % 60;

          XSetForeground(display,gc, colors[ v_arr[index] + 1 ] );

          for(n = 1; n <= number_pixels; n++)
          {
           if( pix_count  ==  60)
             {
              start_pos_x = start_pos_x -60 ;
              start_pos_y = start_pos_y + 1 ;
              pix_count =0;
             }

          XDrawPoint(display, can_xwin, gc, start_pos_x, start_pos_y);
          pix_count++;
          start_pos_x++;
```

```
                    } /* end of n loop */

                   } /* end of index loop */

                 } /* end of count ne 0 */


            else if(count ==1)
               {
                XSetForeground(display,gc, colors[v_arr[0] +1 ] );
                XFillRectangle(display, can_xwin, gc,  start_pos_x ,start_pos_y , 60 , 60);
               }
            else
               {
                XSetForeground(display,gc, colors[BLACK] );
                XFillRectangle(display, can_xwin, gc,  start_pos_x ,start_pos_y , 60 , 60);
               }


          }   /* y loop */


       }   /* x loop */


   }
}

/* end of program */
```

```
/*****************************************************************************/
/* This is the program graph_kohonen.c that integrates the neural net and    */
/* the graphical user interface. This program has the Xview code to draw the*/
/* the frames, panels, buttons etc., to interact with the user for    input */
/* and later run the  neural net. This program can be used to start training*/
/* continue training, and testing of the neural net. The display of the phono*/
/* -topic maps from within this program is available as an option. The code */
/* for doing this is present separately in the kohonen_map.c program and has*/
/* not been incorporated in this program.                                    */
/*****************************************************************************/

#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/cms.h>
#include <xview/notice.h>
#include <xview/notify.h>
#include <math.h>
#include <graph.h>

/**********
#include <kohonen.h>
**********/


/* ********************************GLOBALS ************************** */

Notify_value sigint_func();

/***********
Arrays used for the neural net functions
**********/

        double ***wt;
        double *state;
        double **distance;
        double **local_dist;
        int ***table;
        int **check;
        int **confusion;
        int **node_names;
       int        *layers;

/**********
Variables used in this program
**********/

int Status_flag = START;
int not_confirmed_flag  = FALSE;
int busy_flag = FALSE;
int busy_flag1 = FALSE;
int initialize_flag =0;
int ready_to_exec_flag = FALSE;
int testing_flag = FALSE;
```

```
int initial_neighborhood= 0;
int final_neighborhood = 0;
int input_layer=0;
int x_dimension_output_layer = 0;
int y_dimension_output_layer = 0;
int training_passes =0;
int how_often =0;
int pass_num =0;
int data_samples =0;
double alpha =0.0;
double delta = 0.0;
double decay = 0.0;
double control_neighborhood = 0.0;
char s_alpha[10];
char s_decay[10];
char s_control_neighborhood[10];
char input_wts_filename[20];
char output_wts_filename[20];
char data_filename[20];

FILE *fd_data, *fd_input_wts, *fd_output_wt;

/**********
Xview related declarations for frames, panels, panels items,
cms etc.,
**********/

Panel   panel;
Frame  frame;
Frame start_trainer_frame;
Panel start_trainer_panel;
Frame cont_frame;
Panel cont_panel;
Frame confirm_input_frame;
Panel confirm_input_panel;
Panel_item confirm_1, confirm_2, confirm_3, confirm_4, confirm_5,
           confirm_6, confirm_7, confirm_8;
Display *display;

Cms         cms;

/**********
This structure specifies the rbg values for the colors below
**********/

static Xv_singlecolor colors[] = {
   { 255, 255, 255 }, /* white */
   { 255,   0,   0 }, /* red */
   { 0,   255,   0 }, /* green */
   { 0,     0, 255 }, /* blue */
};
```

```
main(argc, argv)
int argc;
char *argv[];
{


    Rect        rect;
    Event       *event;


/**********
Function declarations
**********/
    void init_data();
    int  go();
    int  show_start_trainer();
    int         show_cont_trainer();
    void test_proc();

    void show_default_proc();
    void quit_proc();

    int  init_neigh_proc();
    int         final_neigh_proc();
    int         input_nodes_proc();
    int         output_x_proc();
    int         output_y_proc();
    int         init_weight();
    int         output_wts_proc();
    int         alpha_proc();
    int         data_proc();
    int         how_often_proc();
    int         training_proc();
    int  toggle_selected();
    int         data_samples_proc();
    int         reset_defaults_proc();
    int  start_exit_proc();
    int  confirm_exit_proc();


    int  cont_read_in_values();
    int  read_go_proc();
    int         input_wts_proc();
    void        init_weight_proc();
    int         cont_exit_proc();
    int         pass_num_proc();

    int         show_confirm_input();


/************************* LOCAL VARIABLES *************************/
        int s_num =0;
        int pass_num =0;
```

```
        int outer_loop_control =0;
        int start_pass =0;
        int c_often =0;
        int t_passes =0;
        int total_passes =0;
        int passes =0;
        int i,j,k,l =0;
        int vowel_type=0;
        int vowel_num=0;
        int vowel_class=0;
        int min_out_x =0;
        int min_out_y =0;
        int error =0;
        int max_times =0;
        int max_vowel =-1;
        int row =0 ;
        int column =0 ;
        int neighborhood = 0;
        int total_iterations =0;
        double c_delta =0.0;
        double c_alpha =0.0;
        double c_decay=0.0;
        double c_neighborhood =0.0;
        int start_num =0;
        int input_node =0;
        int input_da[1520][2];
        double input_db[1520][4];

/******************** CALL TO XV_INIT and SET RECTANGLE VALUES **************/

   xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);


   rect.r_top = rect.r_left = 0;
   rect.r_width = 400;
   rect.r_height = 400;

/***************************** COLOR INITIALIZATION ************************/

        cms = xv_create(XV_NULL, CMS,
                CMS_CONTROL_CMS, TRUE,
                CMS_SIZE, 4 + CMS_CONTROL_COLORS,
                CMS_COLORS, colors,
                CMS_CONTROL_CMS,   TRUE,
                NULL);

/*****************************   MAIN FRAME CREATION ************************/

   frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL, "Kohonen's SOF Map",
        FRAME_NO_CONFIRM,     FALSE,
        FRAME_INHERIT_COLORS,         TRUE,
        FRAME_SHOW_HEADER,  TRUE,
```

```
        FRAME_SHOW_FOOTER,  TRUE,
        NULL);

    frame_set_rect(frame, &rect );

/*************************** PANEL CREATION ************************/

    panel = (Panel)xv_create(frame, PANEL,
            WIN_CMS, cms,
                NULL);

    window_fit(panel);

    /* xv_set(canvas_paint_window(panel), NULL); */

/*************************** CREATE PANEL ITEMS ********************/

    xv_create(panel, PANEL_BUTTON,
        XV_X,               120,
        XV_Y,               40,
        PANEL_LABEL_STRING,   "Start Net Training",
        PANEL_NOTIFY_PROC,    show_start_trainer,
        PANEL_CLIENT_DATA,    frame,
        PANEL_ITEM_COLOR,     CMS_CONTROL_COLORS + BLUE,
        NULL);

    xv_create(panel, PANEL_BUTTON,
        XV_X,               120,
        XV_Y,               120,
        PANEL_LABEL_STRING,   "Continue Training ",
        PANEL_NOTIFY_PROC,    show_cont_trainer,
        PANEL_CLIENT_DATA,    frame,
        PANEL_ITEM_COLOR,     CMS_CONTROL_COLORS + BLUE,
        NULL);

    xv_create(panel, PANEL_BUTTON,
        XV_X,               120,
        XV_Y,               200,
        PANEL_LABEL_STRING,   "Test Trained Net ",
        PANEL_NOTIFY_PROC,    show_cont_trainer,
        PANEL_CLIENT_DATA,    frame,
        PANEL_ITEM_COLOR,     CMS_CONTROL_COLORS + BLUE,
        NULL);

    xv_create(panel, PANEL_BUTTON,
        XV_X,               120,
        XV_Y,               280,
        PANEL_LABEL_STRING,   "Change Defaults ",
        PANEL_NOTIFY_PROC,    show_default_proc,
        PANEL_CLIENT_DATA,    frame,
        PANEL_ITEM_COLOR,     CMS_CONTROL_COLORS + BLUE,
        NULL);
```

```
    xv_create(panel, PANEL_BUTTON,
        XV_X,                   10,
        XV_Y,                   320,
        PANEL_LABEL_STRING,     "Help",
        PANEL_NOTIFY_PROC,      show_default_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + BLUE,
        NULL);

    xv_create(panel, PANEL_BUTTON,
        XV_X,                   340,
        XV_Y,                   320,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      quit_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + BLUE,
        NULL);

/**********
Each of the above has a panel item type (i.e., button), the co-ordinates where
it is to be placed, the label for the button and the callback procedure to be
used when this button is depressed
**********/

        window_fit(panel);

        init_data();



/***************************** START TRAINER BEGINS ************************/

/**********
Create the frame that is to be displayed when the start_trainer button is
depressed, Notice that this frame is a command frame.
**********/

  start_trainer_frame = (Frame)xv_create(frame,FRAME_CMD,
                        XV_WIDTH, 550,
                        XV_HEIGHT,400,
                        XV_X, 400,
                        XV_Y, 0,
                        FRAME_INHERIT_COLORS, TRUE,
                        FRAME_LABEL, "Kohonen Training",
                        FRAME_CMD_PUSHPIN_IN, TRUE,
                        FRAME_DONE_PROC, start_exit_proc,
                        NULL);

/**********
get default panel
**********/
    start_trainer_panel = (Panel)xv_get(start_trainer_frame,FRAME_CMD_PANEL);
```

```
/**********
Set the cms for the panel
***********/


  xv_set( start_trainer_panel, WIN_CMS, cms, NULL);

/***********
create the buttons and other panel items in the panel. The process is the same
as before. The panel items that have been created are self-explanatory
***********/

  xv_create(start_trainer_panel, PANEL_CHOICE,
      PANEL_LABEL_STRING,     "Connection Weights",
      PANEL_CHOICE_STRINGS, "Initialize with Random Weights",
                            "Initialize with Same Weights",
                             NULL,
      PANEL_VALUE,              initialize_flag,
      PANEL_NOTIFY_PROC,      init_weight_proc,
      PANEL_CLIENT_DATA,      frame,
      PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + BLUE,
      NULL);

  xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
      XV_X,                   4,
      XV_Y,                   40,
      PANEL_LABEL_STRING,     "Initial Neighborhood ",
      PANEL_VALUE,            initial_neighborhood,
      PANEL_NOTIFY_PROC,      init_neigh_proc,
      PANEL_CLIENT_DATA,      frame,
      PANEL_MIN_VALUE,        2,
      PANEL_MAX_VALUE,        10,
      PANEL_VALUE_DISPLAY_LENGTH,       2,
      NULL);

   xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
      XV_X,                   4,
      XV_Y,                   70,
      PANEL_LABEL_STRING,     "Final Neighborhood           ",
      PANEL_VALUE,            final_neighborhood,
      PANEL_NOTIFY_PROC,      final_neigh_proc,
      PANEL_CLIENT_DATA,      frame,
      PANEL_MIN_VALUE,        1,
      PANEL_MAX_VALUE,        10,
      PANEL_VALUE_DISPLAY_LENGTH,       2,
      NULL);


   xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
      XV_X,                   4,
      XV_Y,                   100,
      PANEL_LABEL_STRING,     "Number of Input Features",
      PANEL_VALUE,            input_layer,
```

```
        PANEL_NOTIFY_PROC,      input_nodes_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        1,
        PANEL_MAX_VALUE,        16,
        PANEL_VALUE_DISPLAY_LENGTH,     2,
        NULL);

    xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
        XV_X,                   4,
        XV_Y,                   130,
        PANEL_LABEL_STRING,     "Number of Output Nodes in X Dimension  ",
        PANEL_VALUE,            x_dimension_output_layer,
        PANEL_NOTIFY_PROC,      output_x_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        2,
        PANEL_MAX_VALUE,        100,
        PANEL_VALUE_DISPLAY_LENGTH,     3,
        NULL);

    xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
        XV_X,                   4,
        XV_Y,                   160,
        PANEL_LABEL_STRING,     "Number of Output Nodes in Y Dimension  ",
        PANEL_VALUE,            y_dimension_output_layer,
        PANEL_NOTIFY_PROC,      output_y_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        2,
        PANEL_MAX_VALUE,        100,
        PANEL_VALUE_DISPLAY_LENGTH,     3,
        NULL);

    xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
        XV_X,                   4,
        XV_Y,                   190,
        PANEL_LABEL_STRING,     "Number of Training Passes over data File ",
        PANEL_VALUE,            training_passes,
        PANEL_NOTIFY_PROC,      training_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        1,
        PANEL_MAX_VALUE,        100000,
        PANEL_VALUE_DISPLAY_LENGTH,     9,
        NULL);

    xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
        XV_X,                   4,
        XV_Y,                   220,
        PANEL_LABEL_STRING,     "Number of Data Samples in the data File ",
        PANEL_VALUE,            data_samples,
        PANEL_NOTIFY_PROC,      data_samples_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        1,
        PANEL_MAX_VALUE,        2000,
        PANEL_VALUE_DISPLAY_LENGTH,     5,
```

```
        NULL);

    xv_create(start_trainer_panel, PANEL_NUMERIC_TEXT,
        XV_X,                   4,
        XV_Y,                   250,
        PANEL_LABEL_STRING,     "How Often Do you wish to see the update ",
        PANEL_VALUE,            how_often,
        PANEL_NOTIFY_PROC,      how_often_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_MIN_VALUE,        1,
        PANEL_MAX_VALUE,        50000,
        PANEL_VALUE_DISPLAY_LENGTH,     7,
        NULL);

    xv_create(start_trainer_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   280,
        PANEL_LABEL_STRING,     "Data Filename",
        PANEL_VALUE,            data_filename,
        PANEL_NOTIFY_PROC,      data_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     12,
        NULL);

    xv_create(start_trainer_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   310,
        PANEL_LABEL_STRING,     "Output Weights Filename",
        PANEL_VALUE,            output_wts_filename,
        PANEL_NOTIFY_PROC,      output_wts_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     12,
        NULL);

    xv_create(start_trainer_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   340,
        PANEL_LABEL_STRING,     "Alpha value",
        PANEL_VALUE,            s_alpha,
        PANEL_NOTIFY_PROC,      alpha_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     8,
        NULL);

/**********************************START TRAINER ENDS ************************/

/**********************************CONT TRAINER STARTS ************************/

/***********
This where the frame associated with  the continue_training button is created.
As before this frame is displayed when the Continue Training button is depressed
This is also a command frame
```

```
**********/

    cont_frame = (Frame)xv_create(frame,FRAME_CMD,
                        XV_WIDTH, 400,
                        XV_HEIGHT,400,
                        XV_X, 0,
                        XV_Y, 400,
                        FRAME_LABEL, "Continue Training/ Testing",
                        FRAME_CMD_PUSHPIN_IN, TRUE,
                        FRAME_DONE_PROC, cont_exit_proc,
                        NULL);

/**********
Get the default panel
**********/

    cont_panel = (Panel)xv_get(cont_frame,FRAME_CMD_PANEL);

/**********
Create the panel items in the panel
**********/

   xv_create(cont_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   50,
        PANEL_LABEL_STRING,     "Data Filename",
        PANEL_VALUE,            default_data_filename,
        PANEL_NOTIFY_PROC,      data_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     12,
        NULL);

   xv_create(cont_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   150,
        PANEL_LABEL_STRING,     "Input Weights Filename  ",
        PANEL_VALUE,            default_input_wts_filename,
        PANEL_NOTIFY_PROC,      input_wts_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     12,
        NULL);

   xv_create(cont_panel, PANEL_TEXT,
        XV_X,                   4,
        XV_Y,                   250,
        PANEL_LABEL_STRING,     "Output Weights Filename",
        PANEL_VALUE,            default_output_wts_filename,
        PANEL_NOTIFY_PROC,      output_wts_proc,
        PANEL_CLIENT_DATA,      frame,
        PANEL_VALUE_DISPLAY_LENGTH,     12,
        NULL);

   xv_create(cont_panel, PANEL_BUTTON,
```

```
        XV_X,                   4,
        XV_Y,                   330,
        PANEL_LABEL_STRING,     "Confirm Parameters",
        PANEL_NOTIFY_PROC,      show_confirm_input,
        PANEL_CLIENT_DATA,      frame,
        NULL);

/**********************************CONT TRAINER ENDS ***********************/

/**********************************CONFIRM INPUT STARTS*********************/

/***********
This is where the command frame for "Confirm Parameters" button is created
***********/


    confirm_input_frame = (Frame)xv_create(frame,FRAME_CMD,
                        XV_WIDTH, 450,
                        XV_HEIGHT,400,
                        XV_X, 410,
                        XV_Y, 400,
                        FRAME_LABEL, "Confirm Values",
                        FRAME_CMD_PUSHPIN_IN, TRUE,
                        FRAME_DONE_PROC, confirm_exit_proc,
                        NULL);


/***********
get the default panel
***********/
    confirm_input_panel = (Panel) xv_get(confirm_input_frame, FRAME_CMD_PANEL);


/***********
create the panel items
***********/

    confirm_1 = xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
                XV_X,                   10,
                XV_Y,                   40,
                PANEL_LABEL_STRING,     "Initial Neighborhood ",
                PANEL_VALUE,            initial_neighborhood,
                PANEL_NOTIFY_PROC,      init_neigh_proc,
                PANEL_CLIENT_DATA,      frame,
                PANEL_MIN_VALUE,        2,
                PANEL_MAX_VALUE,        10,
                PANEL_VALUE_DISPLAY_LENGTH,     2,
                NULL);

  confirm_2=    xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
                XV_X,                   10,
                XV_Y,                   80,
                PANEL_LABEL_STRING,     "Final Neighborhood           ",
```

```
            PANEL_VALUE,            final_neighborhood,
            PANEL_NOTIFY_PROC,      final_neigh_proc,
            PANEL_CLIENT_DATA,      frame,
            PANEL_MIN_VALUE,        1,
            PANEL_MAX_VALUE,        10,
            PANEL_VALUE_DISPLAY_LENGTH,     2,
            NULL);


confirm_3 = xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
            XV_X,                   10,
            XV_Y,                   120,
            PANEL_LABEL_STRING,     "Number of Training Passes over data File ",
            PANEL_VALUE,            training_passes,
            PANEL_NOTIFY_PROC,      training_proc,
            PANEL_CLIENT_DATA,      frame,
            PANEL_MIN_VALUE,        1,
            PANEL_MAX_VALUE,        100000,
            PANEL_VALUE_DISPLAY_LENGTH,     9,
            NULL);

confirm_4=   xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
            XV_X,                   10,
            XV_Y,                   160,
            PANEL_LABEL_STRING,     "Number of Data Samples in the data File ",
            PANEL_VALUE,            data_samples,
            PANEL_NOTIFY_PROC,      data_samples_proc,
            PANEL_CLIENT_DATA,      frame,
            PANEL_MIN_VALUE,        1,
            PANEL_MAX_VALUE,        2000,
            PANEL_VALUE_DISPLAY_LENGTH,     5,
            NULL);

confirm_5=  xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
            XV_X,                   10,
            XV_Y,                   200,
            PANEL_LABEL_STRING,     "The Current Pass Number is",
            PANEL_VALUE,            pass_num,
            PANEL_NOTIFY_PROC,      pass_num_proc,
            PANEL_CLIENT_DATA,      frame,
            PANEL_MIN_VALUE,        0,
            PANEL_MAX_VALUE,        100000,
            PANEL_VALUE_DISPLAY_LENGTH,     7,
            NULL);

confirm_6=  xv_create(confirm_input_panel, PANEL_NUMERIC_TEXT,
            XV_X,                   10,
            XV_Y,                   240,
            PANEL_LABEL_STRING,     "How Often would you to see the update",
            PANEL_VALUE,            how_often,
            PANEL_NOTIFY_PROC,      how_often_proc,
            PANEL_CLIENT_DATA,      frame,
            PANEL_MIN_VALUE,        1,
```

```
                PANEL_MAX_VALUE,        50000,
                PANEL_VALUE_DISPLAY_LENGTH,     7,
                NULL);

    confirm_7=  xv_create(confirm_input_panel, PANEL_TEXT,
                XV_X,                   10,
                XV_Y,                   280,
                PANEL_LABEL_STRING,     "Alpha value",
                PANEL_VALUE,            s_alpha,
                PANEL_NOTIFY_PROC,      alpha_proc,
                PANEL_CLIENT_DATA,      frame,
                PANEL_VALUE_DISPLAY_LENGTH,     8,
                NULL);

    confirm_8=  xv_create(confirm_input_panel, PANEL_CHECK_BOX,
                XV_X,                   10,
                XV_Y,                   310,
                PANEL_CHOOSE_ONE,       TRUE,
                PANEL_LABEL_STRING,     "Testing ?",
                PANEL_CHOICE_STRINGS,"No", "Yes",NULL,
                PANEL_VALUE,            testing_flag,
                PANEL_NOTIFY_PROC,      test_proc,
                PANEL_CLIENT_DATA,      frame,
                NULL);

/**********************************CONFIRM INPUT ENDS ********************/


/***********
use this to handle signals
***********/

        notify_set_signal_func(frame, sigint_func, SIGINT, NOTIFY_SYNC);

        display = (Display *) xv_get(frame, XV_DISPLAY);

        xv_main_loop(frame);

        XFlush(display);


/************
The notifier has been stopped by the call back routine and now the dispatcher
takes over
*************/

/***********
This is the heart of the control stuff for the neural net The control.c program
(that has the kohonen function) is present below.
***********/


        total_passes = training_passes;
```

```
        passes = pass_num;


        outer_loop_control =layers[Output_X]*layers[Output_Y] * total_passes;
        total_iterations =outer_loop_control * data_samples ;


        c_alpha = alpha;
        c_delta = delta;
        t_passes = total_passes;
        c_often = how_often;
       start_pass = passes;
        c_decay = decay;
        c_neighborhood = control_neighborhood;

        if(c_neighborhood ==0.0)
        {
       c_decay =exp(log((float)final_neighborhood/(float)initial_neighborhood)
                                                  /(float)total_iterations);
       c_neighborhood= (double)initial_neighborhood;

       c_delta = exp(log( 0.0000001 /alpha) / total_iterations);

      }

        for(i=0; i<1520;i++)
        {
         fscanf(fd_data, "%d %d ", &input_da[i][0], &input_da[i][1]);

          for(j=0;j<4;j++)

          {
           fscanf(fd_data, "%le", &input_db[i][j]);
          }
         }

      XFlush(display);

 for(pass_num=start_pass;pass_num <outer_loop_control;pass_num++)
   {

/*********
This what flushes events to the server...The more often you do this, the better
is the response.
*********/
           notify_dispatch();
           XFlush(display);

          for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
          {

               s_num = input_da[vowel_num][0];
               vowel_class = input_da[vowel_num][1];
```

```
            for(input_node=Zero;input_node<layers[Input];input_node++)
                {
                 state[input_node] = input_db[vowel_num][input_node+1];
                }


        calculate_distance(layers,&min_out_x,&min_out_y);

        if(testing_flag == FALSE)
           adjust_wt(layers,&c_alpha,&c_neighborhood,&min_out_x,&min_out_y);

        c_neighborhood = c_neighborhood * c_decay ;
         c_alpha = c_alpha * c_delta;


        }

         notify_dispatch();
         XFlush(display);

       if((pass_num % how_often) == 0)
       {
          fprintf(stderr,"control %d\n",pass_num);

           for(vowel_type=0;vowel_type<Vowel_type_max;vowel_type++)
            {
             for(row =Zero;row <layers[Output_X];row++)
              {
               for(column=Zero;column<layers[Output_Y];column++)
                {

                    table[vowel_type][row][column] = 0;
                  }
              }
             }


          for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
           {

                    s_num = input_da[vowel_num][0];
                    vowel_class = input_da[vowel_num][1];

                    for(input_node=Zero;input_node<layers[Input];input_node++)
                    {
                     state[input_node] = input_db[vowel_num][input_node+1];
                     }


                    calculate_distance( layers,&min_out_x,&min_out_y);
                    table[vowel_class][min_out_x][min_out_y]++;
             }
```

```
for(row = Zero; row <layers[Output_X];row++)
   {
     for(column=Zero;column<layers[Output_Y];column++)
      {

          max_times =0;
          max_vowel=-1;


       for(vowel_type=0;vowel_type<Vowel_type_max;vowel_type++)
        {

          if(table[vowel_type][row][column] >max_times)
          {
                max_times = table[vowel_type][row][column];
                max_vowel = vowel_type;
           }

        }
       node_names[row][column] = max_vowel;

      }
    }



 for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
    {
        s_num = input_da[vowel_num][0];
        vowel_class = input_da[vowel_num][1];


         for(input_node=Zero;input_node<layers[Input];input_node++)
           {
            state[input_node] = input_db[vowel_num][input_node+1];
           }


       calculate_distance(layers,&min_out_x,&min_out_y);

          if(vowel_class != node_names[min_out_x][min_out_y] )
         error++;

       confusion[vowel_class][ node_names[min_out_x][min_out_y]]++;

    }


 map_to_table(&pass_num,&c_often,layers);
 confus(layers, error, data_samples, pass_num);


write_final_wts(fd_output_wt,layers,&initial_neighborhood,&final_neighborhood, &c_alpha,
```

```
&c_delta,&c_neighborhood,&c_decay,&how_often, &data_samples, &pass_num,&total_passes);

            fflush(fd_output_wt);
            }

    }


     write_final_wts(fd_output_wt,layers,&initial_neighborhood,&final_neighborhood, &c_alpha,
&c_delta,&c_neighborhood,&c_decay,&how_often, &data_samples, &pass_num,&total_passes);



}
/**************end of main loop */


/*********
init_data
*********/

void
init_data()
{

        initialize_flag =default_initialize_flag;
        initial_neighborhood= default_initial_neighborhood;
        final_neighborhood = default_final_neighborhood;
        input_layer=default_input_layer_nodes;
        x_dimension_output_layer = default_x_dimension_output_layer;
        y_dimension_output_layer = default_y_dimension_output_layer;
        training_passes =default_training_passes;
        how_often =default_how_often;
        pass_num =default_pass_num;
        data_samples =default_data_samples;
        strcpy(s_alpha ,default_alpha_value);
        alpha = atof(s_alpha);
        strcpy(s_decay ,default_decay_value);
        decay = atof(s_decay);
        strcpy(s_control_neighborhood ,default_control_neighborhood);
        control_neighborhood = atof(s_control_neighborhood);
        strcpy(input_wts_filename,default_input_wts_filename);
        strcpy(output_wts_filename, default_output_wts_filename);
        strcpy(data_filename, default_data_filename);


}



/*********
show start trainer frame
*********/
```

```
int
show_start_trainer(item,event)
Frame item;
Event *event;
{
        int answer =0;

/********
put up a notice asking for parameters
*********/

    if(busy_flag ==FALSE && busy_flag1 == FALSE)
     {

        busy_flag = TRUE;
        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "Enter the values for the parameters",
                                "in the next frame and CLICK on the",
                                "PUSHPIN on top left when done",NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
        xv_set(start_trainer_frame, XV_SHOW, TRUE, NULL);

     }

    else
        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "You are using another frame", NULL,
                                NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);

}

/*********
start trainer call back routines
*********/


int
selected(item, value, event)
Panel_item item;
int value;
Event *event;
{
   char buf[32];
   Frame frame = xv_get(item, PANEL_CLIENT_DATA);
   if (event_id(event) == MS_LEFT) {
        sprintf(buf, "\"%s\" selected",
            xv_get(item, PANEL_CHOICE_STRING, panel_get_value(item)));
        xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
```

```
        return XV_OK;
    }
    return XV_ERROR;
}


int
init_neigh_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int value;
    int answer;
    Frame local_frame;
    Panel local_panel;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    initial_neighborhood = xv_get(item,PANEL_VALUE);

    if( initial_neighborhood == 2)
     {
         notice_proc(Minimum);
         return PANEL_NONE;
      }
     else if(initial_neighborhood <2)
     {
         notice_proc(LtMinimum);
         initial_neighborhood = default_initial_neighborhood;
        xv_set(item, PANEL_VALUE, initial_neighborhood,NULL);
         return PANEL_NEXT;

      }
     else if(initial_neighborhood >10)
     {
         notice_proc(GtMaximum);
         initial_neighborhood = default_initial_neighborhood;
        xv_set(item, PANEL_VALUE, initial_neighborhood,NULL);
         return PANEL_NEXT;

      }
     else if(initial_neighborhood ==10)
     {
         notice_proc(Maximum);
         return PANEL_NONE;

      }
    else
      return PANEL_NEXT;

}

final_neigh_proc(item, event)
```

```
Panel_item item;
Event *event;
{
   char buf[132];
   int value;
   int answer =0;

   Frame frame = xv_get(item, PANEL_CLIENT_DATA);
   final_neighborhood = xv_get(item, PANEL_VALUE);


   if(final_neighborhood  == 1 )
   {
        notice_proc(Minimum);
        PANEL_NONE;
   }
   else if(final_neighborhood  == 10 )
   {
        notice_proc(Maximum);
        PANEL_NONE;
   }
    else if(final_neighborhood <1)
    {
        notice_proc(LtMinimum);
        final_neighborhood = default_final_neighborhood;
       xv_set(item, PANEL_VALUE, final_neighborhood,NULL);
        return PANEL_NONE;

     }
    else if(final_neighborhood >10)
    {
        notice_proc(GtMaximum);
        final_neighborhood = default_final_neighborhood;
       xv_set(item, PANEL_VALUE, final_neighborhood,NULL);
        return PANEL_NONE;

     }

   else
        return PANEL_NEXT;
}


data_samples_proc(item, event)
Panel_item item;
Event *event;
{
   char buf[132];
   int value;
   int answer =0;

   Frame frame = xv_get(item, PANEL_CLIENT_DATA);
   data_samples = xv_get(item, PANEL_VALUE);
```

```
    if(data_samples  == 1 )
    {
         notice_proc(Minimum);
         PANEL_NONE;
    }
    else if(data_samples  == 2000 )
    {
         notice_proc(Maximum);
         PANEL_NONE;
    }
     else if(data_samples <1)
     {
         notice_proc(LtMinimum);
         data_samples = default_data_samples;
        xv_set(item, PANEL_VALUE, data_samples,NULL);
         return PANEL_NONE;

      }
     else if(data_samples > 2000 )
     {
         notice_proc(GtMaximum);
         data_samples = default_data_samples;
        xv_set(item, PANEL_VALUE, data_samples,NULL);
         return PANEL_NONE;
     }
    else
         return PANEL_NEXT;
}

input_nodes_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int answer=0;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    input_layer = xv_get(item,PANEL_VALUE);


    if( input_layer < 1 )
     {
       notice_proc(LtMinimum);
          input_layer = default_input_layer_nodes;
       xv_set(item, PANEL_VALUE, input_layer,NULL);
       return PANEL_NONE;
     }
    if( input_layer == 1 )
     {
       notice_proc(Minimum);
       return PANEL_NONE;
     }
```

```
    else if( input_layer > 16 )
     {
       notice_proc(GtMaximum);
          input_layer = default_input_layer_nodes;
       xv_set(item, PANEL_VALUE, input_layer,NULL);
       return PANEL_NONE;
      }

    else if( input_layer == 15 )
     {
       notice_proc(Maximum);
       return PANEL_NONE;
     }

    else
         return  PANEL_NEXT;


}

output_x_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int value;
    int answer;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    x_dimension_output_layer = xv_get(item,PANEL_VALUE);

    if( x_dimension_output_layer== 2 )
     {
         notice_proc(Minimum);
         return PANEL_NONE;
     }
     else if(x_dimension_output_layer ==  100)
     {
         notice_proc(Maximum);
         return PANEL_NONE;
     }
    else if( x_dimension_output_layer < 2 )
     {
       notice_proc(LtMinimum);
                x_dimension_output_layer = default_x_dimension_output_layer;
              xv_set(item, PANEL_VALUE, x_dimension_output_layer,NULL);
              return PANEL_NONE;
             }
            else if( x_dimension_output_layer > 100 )
             {
               notice_proc(GtMaximum);
                 x_dimension_output_layer = default_x_dimension_output_layer;
      xv_set(item, PANEL_VALUE, x_dimension_output_layer,NULL);
```

```
      return PANEL_NONE;
     }
    else
        return PANEL_NEXT;

}

output_y_proc(item, event)
Panel_item item;
Event *event;
{
   char buf[132];
   int value;
   int answer;

   Frame frame = xv_get(item, PANEL_CLIENT_DATA);
   y_dimension_output_layer = xv_get(item,PANEL_VALUE);

   if( y_dimension_output_layer== 2 )
    {
        notice_proc(Minimum);
        return PANEL_NONE;
    }
    else if(y_dimension_output_layer ==  100)
    {
        notice_proc(Maximum);
        return PANEL_NONE;
    }
   else if( y_dimension_output_layer < 2 )
    {
     notice_proc(LtMinimum);
        y_dimension_output_layer = default_y_dimension_output_layer;
     xv_set(item, PANEL_VALUE, y_dimension_output_layer,NULL);
     return PANEL_NONE;
     }
   else if( y_dimension_output_layer > 100 )
    {
     notice_proc(GtMaximum);
        y_dimension_output_layer = default_y_dimension_output_layer;
     xv_set(item, PANEL_VALUE, y_dimension_output_layer,NULL);
     return PANEL_NONE;
     }
    else
        return PANEL_NEXT;

}


training_proc(item, event)
Panel_item item;
Event *event;
{
   char buf[132];
```

```
    int value;
    int answer;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);

    training_passes = xv_get(item, PANEL_VALUE);

    if( training_passes== 1)
     {
         notice_proc(Minimum);
         return PANEL_NONE;
     }
    else if( training_passes== 100000)
     {
         notice_proc(Maximum);
         return PANEL_NONE;
     }
    else if( training_passes > 100000)
     {
         notice_proc(GtMaximum);
         training_passes = default_training_passes;
       xv_set(item, PANEL_VALUE, training_passes,NULL);
         return PANEL_NONE;
     }
    else if( training_passes < 1)
     {
         notice_proc(LtMinimum);
         training_passes = default_training_passes;
       xv_set(item, PANEL_VALUE, training_passes,NULL);
         return PANEL_NONE;
     }

    else
         return PANEL_NEXT;
}

how_often_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int answer;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);

    how_often = xv_get(item,PANEL_VALUE);

    if( how_often== 1)
      {
         notice_proc(Minimum);
         return PANEL_NONE;
      }
    else if(  how_often ==  100000)
```

```
    {
        notice_proc(Maximum);
        return PANEL_NONE;
    }
    else if( how_often > 100000)
    {
        notice_proc(GtMaximum);
        how_often = default_how_often;
      xv_set(item, PANEL_VALUE, how_often,NULL);
        return PANEL_NONE;
    }
    else if( how_often < 1)
    {
        notice_proc(LtMinimum);
        how_often = default_how_often;
      xv_set(item, PANEL_VALUE, how_often,NULL);
        return PANEL_NONE;
    }

    else
     return PANEL_NEXT;
}

data_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int answer;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    strcpy(data_filename, xv_get(item,PANEL_VALUE));

     if(strcmp(data_filename,"") <= 0)
     {
         strcpy(data_filename, default_data_filename);
         answer = notice_prompt( xv_get(item, PANEL_PARENT_PANEL), NULL,
                                 NOTICE_MESSAGE_STRINGS,
                                 "You have specified no datafile",
                                 "The default is being used",
                                  NULL,
                                 NOTICE_BUTTON_YES, "OK",
                                 NULL);

    xv_set(item, PANEL_VALUE, data_filename,NULL);

     }

         if((fd_data = fopen(data_filename,"r")) == NULL)
         {
                 answer = notice_prompt( panel, NULL,
                 NOTICE_MESSAGE_STRINGS, " File", data_filename,
                 " can't be opened or does not exist",
```

```
                                NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
        return PANEL_NONE;
       }

         else
          return PANEL_NEXT;
}

output_wts_proc(item, event)
Panel_item item;
Event *event;
{
   char buf[132];
   int answer;

   Frame frame = xv_get(item, PANEL_CLIENT_DATA);

   strcpy(output_wts_filename, xv_get(item,PANEL_VALUE));
   if(strcmp(output_wts_filename,"") <= 0)
    {
        strcpy(output_wts_filename, default_output_wts_filename);
        answer = notice_prompt( xv_get(item, PANEL_PARENT_PANEL), NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "You have specified no filename",
                                "The default is being used",
                                 NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
   xv_set(item, PANEL_VALUE, output_wts_filename,NULL);

    }
        if((fd_output_wt = fopen(output_wts_filename,"w")) == NULL)
        {
                answer = notice_prompt( panel, NULL,
                NOTICE_MESSAGE_STRINGS, " File", output_wts_filename,
                " can't be opened or does not exist",
                                 NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
      return PANEL_NONE;
      }

   else
        PANEL_NEXT;

}

alpha_proc(item, event)
Panel_item item;
Event *event;
{
```

```
    char buf[132];
    char  value[20];
    int answer;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    strcpy(value, xv_get(item,PANEL_VALUE));
    alpha = atof(value);
     if(alpha == 0.0)
       {
         answer = notice_prompt( xv_get(item, PANEL_PARENT_PANEL), NULL,
                                 NOTICE_MESSAGE_STRINGS,
                                 "Alpha value not specified or is Zero",
                          " Using Default Alpha ",
                                  NULL,
                                 NOTICE_BUTTON_YES, "OK",
                                 NULL);

         strcpy(s_alpha ,default_alpha_value);
         alpha = atof(s_alpha);
         xv_set(item, PANEL_VALUE, s_alpha,NULL);
       }

    printf("\n Alpha set to = %s\n",value);
    printf("\n Alpha in float is  = %f\n",alpha);

    return PANEL_NONE;

}


void
init_weight_proc(item,value,event)
Panel_item item;
int value;
Event *event;
{
    char buf[132];
    int choice =0;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    initialize_flag = (int) xv_get(item, PANEL_VALUE);

}

/********
This is the callback routine that specifies whether Testing or Training is being
carried out.
*********/


void
test_proc(item, event)
Frame item;
```

```
Event *event;
{
  int choice =0;
   choice = (int) xv_get(item, PANEL_VALUE);
   if(choice == 0)
    {
      testing_flag = TRUE;
    }

}
void
show_default_proc(item, event)
Frame item;
Event *event;
{
        printf("\nPanel Item %s selected\n",xv_get(item,PANEL_CLIENT_DATA));
        printf("\n Not yet Included in this program \n");

}


/********
This is the callback routine that accepts the inputs for a new run,
validates it and if all parameters are present, stops the notifier.
*********/

int
start_exit_proc(local_frame)
Frame local_frame;
{

  int answer =0;

/*  busy_flag = FALSE; */

  xv_set(local_frame, XV_SHOW, FALSE,  NULL);


/***************************** Check Start STuff ******************/

   if(initial_neighborhood ==0)
        initial_neighborhood = default_initial_neighborhood;

   if(final_neighborhood ==0)
        final_neighborhood = default_final_neighborhood;

   if(input_layer ==0)
        input_layer = default_input_layer_nodes;

   if(x_dimension_output_layer ==0)
        x_dimension_output_layer = default_x_dimension_output_layer;

   if(y_dimension_output_layer ==0)
```

```
    if(training_passes ==0)
      training_passes = default_training_passes;


    if(how_often ==0)
      how_often = default_how_often;

    if(data_samples ==0)
      data_samples = default_data_samples;



    /* malloc for node */

     if(!(layers = (int *) malloc(Three * sizeof(int))))
       {
          answer = notice_prompt( panel, NULL,
                                  NOTICE_MESSAGE_STRINGS,
                                  "Out of memory ",
                                  "QUITTING APPLICATION...",
                                   NULL,
                                  NOTICE_BUTTON_YES, "OK",
                                  NULL);

        fprintf(stderr,"\nOut of memory\n");
        exit(0);
      }


     layers[Input]      = input_layer;
     layers[Output_X]   = x_dimension_output_layer;
     layers[Output_Y]   = y_dimension_output_layer;

/*********************************FILE CHECK ********************************/


    if(fd_output_wt  == NULL)
      {
          if((fd_output_wt = fopen(data_filename,"w")) == NULL)
           {
            answer = notice_prompt( panel , NULL,
                                  NOTICE_MESSAGE_STRINGS,
                        " File", output_wts_filename,
                                  " can't be opened or does not exist",
                                  "QUITTING APPLICATION...",
                                   NULL,
                                  NOTICE_BUTTON_YES, "OK",
                                  NULL);
          }
        printf("\n File %s can't be opened \n",output_wts_filename);
          exit(0);
```

```
    }

   if(fd_data == NULL)
    {
        if((fd_data = fopen(data_filename,"r")) == NULL)
         {
              answer = notice_prompt( panel, NULL,
              NOTICE_MESSAGE_STRINGS, " File", input_wts_filename,
              " can't be opened or does not exist",
                              "QUITTING APPLICATION...",
                               NULL,
                              NOTICE_BUTTON_YES, "OK",
                              NULL);
         }
      printf("\n File %s can't be opened \n",output_wts_filename);
      exit(0);

    }


      mem_alloc(layers);

      initialize(layers, &initialize_flag );

      ready_to_exec_flag = TRUE;

      notify_stop();

/************
This where the xv_main_loop stops and flushing and dispatching is started
once every so often. This is to  allow the reading of events and continuation
of the computation  at the same time
*************/


}

int
cont_exit_proc(local_frame)
Frame local_frame;
{
  int answer =0;
  int c, i=0;
  char s_alpha[20];


  xv_set(local_frame, XV_SHOW, FALSE,   NULL);
  cont_read_in_values();
  notify_stop();
}


cont_read_in_values()
```

```
cont_read_in_values()
{


  int answer =0;
  int  i=0;

 if(not_confirmed_flag == FALSE)

}

/********
This is the callback routine that displays the notice when the values have been
read in from the input weights file
********/

int
show_confirm_input(item,event)
Frame item;
Event *event;
{
        int answer =0;

        if(busy_flag1 == TRUE)
        {

        answer = notice_prompt(panel, NULL,
                 NOTICE_MESSAGE_STRINGS,
                "Another similar frame is in use",  NULL,
                 NOTICE_BUTTON_YES, "ok",
                NULL);

        }

        else
        {
        answer = notice_prompt(panel, NULL,
                 NOTICE_MESSAGE_STRINGS,
                "These are the values read in from the input weights file",
                "Please change them if you wish to and when done CLICK",
                "on the PUSHPIN",  NULL,
                 NOTICE_BUTTON_YES, "ok",
                NULL);

        xv_set(confirm_input_frame, XV_SHOW, TRUE, NULL);
        busy_flag1 = TRUE;
        cont_read_in_values();
        not_confirmed_flag = TRUE;
        }

}

/********
```

```
********/

int
confirm_exit_proc(local_frame)
Frame local_frame;
{
  busy_flag1 = FALSE;

    initial_neighborhood =xv_get(confirm_1, PANEL_VALUE);
    final_neighborhood = xv_get(confirm_2, PANEL_VALUE );
    training_passes = xv_get(confirm_3, PANEL_VALUE);
    data_samples = xv_get(confirm_4, PANEL_VALUE);
    pass_num = xv_get(confirm_5, PANEL_VALUE);
    how_often = xv_get(confirm_6, PANEL_VALUE);
    alpha = atof(xv_get(confirm_7, PANEL_VALUE));
    control_neighborhood = 0.0;
    decay =0.0;
    delta =0.0;

   xv_set(local_frame, XV_SHOW, FALSE,  NULL);
}

/********
This is the callback routine that handles the signals
********/

Notify_value
sigint_func(client, sig, when)
Notify_client client;
int sig;
{
        xv_destroy_safe(frame);
        return NOTIFY_DONE;
}

/********
This is the callback routine that quits the application
********/
void
quit_proc(item, event)
Panel_item item;
Event *event;
{
   Frame frame = xv_get(item, PANEL_CLIENT_DATA);
   Panel panel = xv_get(item, PANEL_PARENT_PANEL);

   int answer = notice_prompt(       panel, NULL,
                NOTICE_FOCUS_XY, event_x(event), event_y(event),
                NOTICE_MESSAGE_STRINGS,
                "Are you sure you want to quit ??",
                NULL,
                NOTICE_BUTTON_YES, "YES",
                NOTICE_BUTTON_NO, "NO",
```

```
                NULL);

  if(answer == NOTICE_YES)
   {
   exit(0);
   }

}

/********
This is the procedure for displaying the notices based on the values entered
by the user.
********/

notice_proc(value)
int value;
{
   int answer=0;

        if(value == LtMinimum)
        {
        answer = notice_prompt( panel, NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "This is below the minimum allowable",
                                "number for this variable",
                                "The number will be reset to the default value",
                                "If you need a smaller number ",
                                "Please change the defaults ", NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
        }
        if(value == Minimum)
        {
        answer = notice_prompt( panel, NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "This is the minimum allowable",
                                "number for this variable",
                                "If you need a smaller number ",
                                "Please change the defaults ", NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
        }
        else if (value == GtMaximum)
        {
        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "This is above the maximun allowable",
                                "number for this variable",
                                "The number will be reset to the default value",
                                "If you need a larger number ",
                                "Please change the defaults ", NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
```

```
        }

        else if (value == Maximum)
        {
        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "This is the maximun allowable",
                                "number for this variable",
                                "If you need a larger number ",
                                "Please change the defaults ", NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
        }


}


/********
This is the procedure for displaying the notice when the Contunue Training
button is depressed
********/

show_cont_trainer(item, event)
Frame item;
Event *event;
{

        int answer =0;

  if(busy_flag == FALSE)
   {
        Status_flag = CONTINUE;

        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "Enter the values for the parameters",
                                "in the next frame and CLICK on the",
                                "PUSHPIN on top left when done",NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);

        xv_set(cont_frame, XV_SHOW, TRUE, NULL);
        busy_flag = TRUE;
   }

   else
        answer = notice_prompt( panel,  NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "Another frame is in use",NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
}
```

```
/********
This is the call back procedure for getting the input weights filename
********/

input_wts_proc(item, event)
Panel_item item;
Event *event;
{
   char buf[132];
   int answer =0;

   Frame frame = xv_get(item, PANEL_CLIENT_DATA);

   strcpy(input_wts_filename , xv_get(item,PANEL_VALUE));

   if(strcmp(input_wts_filename,"") <= 0)
    {
        strcpy(input_wts_filename, default_input_wts_filename);
        answer = notice_prompt( xv_get(item, PANEL_PARENT_PANEL), NULL,
                                NOTICE_MESSAGE_STRINGS,
                                "You have specified no datafile",
                                "The default is being used",
                                 NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
   xv_set(item, PANEL_VALUE, input_wts_filename,NULL);


    }
        if((fd_input_wts = fopen(input_wts_filename,"r")) == NULL)
        {
                answer = notice_prompt( panel, NULL,
                NOTICE_MESSAGE_STRINGS, " File", input_wts_filename,
                " can't be opened or does not exist",
                                "QUITTING APPLICATION...",
                                 NULL,
                                NOTICE_BUTTON_YES, "OK",
                                NULL);
      return PANEL_NONE;
     }

   else
   return PANEL_NEXT;
}


/********
This is the call back procedure for getting the current pass number
********/
```

```
pass_num_proc(item, event)
Panel_item item;
Event *event;
{
    char buf[132];
    int value;
    int answer =0;

    Frame frame = xv_get(item, PANEL_CLIENT_DATA);
    final_neighborhood = xv_get(item, PANEL_VALUE);

    if(pass_num  == 0 )
    {
         notice_proc(Minimum);
         PANEL_NONE;
    }
    else if(pass_num  > training_passes )
    {
         notice_proc(GtMaximum);
         pass_num = default_pass_num;
        xv_set(item, PANEL_VALUE, pass_num,NULL);
         PANEL_NEXT;
    }
     else if(pass_num == training_passes)
     {
         notice_proc(Maximum);
         return PANEL_NEXT;

      }

    else
         return PANEL_NEXT;
}
```

```
/*****************************************************************************/
/* These are the constants used in the above program. A lot of them are common*/
/* to the neural net and can be found in the kohonen.h program          */
/*****************************************************************************/

#define gray_width 2
#define gray_height 2

#define WHITE           0
#define RED         1
#define GREEn           2
#define BLUE            3

#define TRUE        1
#define FALSE           0
#define START           0
#define CONTINUE    1
#define Input       0
#define Output_X    1
#define Output_Y    2
#define Three       3
#define default_initialize_flag 0
#define default_initial_neighborhood 3
#define default_final_neighborhood 1
#define default_input_layer_nodes 3
#define default_x_dimension_output_layer 8
#define default_y_dimension_output_layer 8
#define default_training_passes 10
#define default_how_often 1
#define default_data_samples  1520
#define default_pass_num  0
#define default_data_filename "f0_f3.big"
#define default_output_wts_filename  "default.wts"
#define default_input_wts_filename  "saved.wts"
#define default_alpha_value  "0.1"
#define default_control_neighborhood  "0.0"
#define default_decay_value  "0.0"
#define Minimum 0
#define LtMinimum 1
#define Maximum 2
#define GtMaximum 3
```

```
/*************************************************************************/
/* This is the function for adjusting wts. The adjustment is based on the    */
/* alpha value and the proximity of the node to the winning node (node with) */
/* least distance. The distance that is  used for  determining the proximity */
/* is the  Eucledian distance measure. The factor ratio is used to determine */
/* the amount of the update. Closer the node to the winning node, more is the*/
/* change in the weight. A check feature is used to ensure that all nodes in */
/* the grid in that adjustment cycle are updated only once. This funtion uses*/
/* NO WRAP AROUND                                                            */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "kohonen.h"

 extern          double ***wt;
 extern     double *state;
 extern     double **distance;
 extern     double **local_dist;
 extern     int ***table;
 extern     int ***check;

adjust_wt(fd_result,layers,alpha,neighborhood,min_out_x,min_out_y)
FILE  *fd_result;
double *alpha;
int *layers,*min_out_x,*min_out_y;
double *neighborhood;
{
int x = 0;
int y =0;
int x1=0;
int y1=0;
int upper_x = 0;
int upper_y = 0;
int lower_x = 0;
int lower_y = 0;
int input = 0;
int i=0;
int j=0;
int x_row =0;
int y_row =0;
int adjust_neighborhood;
int neighborhood_limit = 0;
double local_alpha = 0.0;
double dist[16][16];
double ratio=0.0;

/*
The nint function is available only on the Suns. If it is not available,
use the following
*/
```

```
  adjust_neighborhood= nint( *neighborhood ) ;

  neighborhood_limit = 2 * ( adjust_neighborhood ) + 1;


/*
Malloc for the dist array. Use this if you wish to after changing the declaration
above

 if(!(dist = (double **) malloc(neighborhood_limit * sizeof(double)) ))
  {
   fprintf(stderr, "\n Out of memory or Allocation failure\n");
   exit(2);
  }
 for(i=0;i<neighborhood_limit;i++)
  {
 if(!(dist[i] = (double *) malloc(neighborhood_limit * sizeof(double) )))
   {
    fprintf(stderr, "\n Out of memory or Allocation failure\n");
    exit(2);
   }
  }
 */


/*
Set all the elements in the array to 0
*/

  for(i=0 ; i<neighborhood_limit;i++)
   {
     for(j=0;j<neighborhood_limit;j++)
      {
         dist[i][j]= 0.0;
      }
   }

/*
The check array keeps track of the weight update and ensures that each
node gets updated only once.
*/

  for(x_row = Zero; x_row<layers[Output_X];x_row++)
  {
   for(y_row=0;y_row<layers[Output_Y];y_row++)
    {
     for(input=Zero;input <layers[Input];input++)
     {
         check[input][x_row][y_row]= 0;
     }
    }
  }
```

```
/*
Compute the distances between the nodes
*/

 for(i=0 ; i<neighborhood_limit;i++)
  {
  for(j=0;j<neighborhood_limit;j++)
   {
       dist[i][j]= sqrt( pow(  ( *neighborhood - (double) i), 2.0)  +
                         pow(  ( *neighborhood - (double) j), 2.0  ) );

   }
  }


/*
Compute the bounds for the update
*/

 local_alpha = *alpha;
 lower_x     = *min_out_x - adjust_neighborhood;
 lower_y     = *min_out_y - adjust_neighborhood;
 upper_x     = *min_out_x + adjust_neighborhood;
 upper_y     = *min_out_y + adjust_neighborhood;

 for(x=lower_x, i=0;x<=upper_x;x++,i++)
 {


/*
This is without wrap around. This takes lesser time but is not as good in
performance as with the wraparound
*/

    if(x <0 || x > (layers[Output_X] -1))
      continue;

    else
      x1 = x;

  for(y = lower_y, j=0;y<= upper_y;y++,j++)
   {

    if(y <0  ||  y > (layers[Output_Y] - 1 ) )
        continue;
    else
      y1 = y;

/*
Compute the ratio as explained in the text. if the distance between the
nodes is greater than the current neighborhood saturate the ratio at 1
Use the weight adjust formula that incorporates the ratio as shown below
*/
```

```
    for(input=Zero;input <layers[Input];input++)
    {
     if( dist[i][j] > *neighborhood)
       ratio = 1.0;
     else
       ratio = dist[i][j]/ *neighborhood;


     if(check[input][x1][y1] ==0)
        {
        wt[input][x1][y1] = wt[input][x1][y1] +
              (1.0 - ratio) * local_alpha * (state[input] - wt[input][x1][y1]);

       check[input][x1][y1]= 1;

       }

    }

  }  /* end of for for y */

 }  /* end of for for x */


} /* end of adjust weights */
```

```
/*************************************************************************/
/* This is the function that runs the kohonen algorithm. The main routine  */
/* passes control for the required number of passes to be executed. This   */
/* calls the other functions to do the tasks of reading data,calculating   */
/* ouptut, adjusting weights, printing the confusion matrix, writing to the */
/* file the weights, printing out the tables                               */
/*************************************************************************/


/*
The difference between this function and the other kohonen function is
that this is used exclusively for testing. When you have a large number of
weight files that you want to test for performance...use this file...
Notice that there is no training involved (distance calculation and weight
adjustment
*/



#include <stdio.h>
#include <math.h>
#include <string.h>
#include <values.h>
#include "kohonen.h"


extern          double ***wt;
extern    double *state;
extern    double **distance;
extern    double **local_dist;
extern    int ***table;
extern    int ***check;
extern          int **confusion;
extern    int **node_names;


kohonen(fd_data,fd_output_wt,layers,initial_neighborhood,final_neighborhood,alpha,
delta, control_neighborhood, decay, how_often, data_samples, passes, total_passes)
FILE *fd_data,*fd_output_wt;
int *layers,initial_neighborhood,final_neighborhood;
int passes,total_passes,how_often, data_samples;
double delta, control_neighborhood, decay;
double alpha;
{

        int s_num =0;
        int pass_num =0;
        int outer_loop_control =0;
        int start_pass =0;
        int c_often =0;
        int t_passes =0;
        int i,j,k,l =0;
```

```
        int vowel_type=0;
        int vowel_num=0;
        int vowel_class=0;
        int error =0;
        int max_vowel = -1;
        int max_times =0;
        int min_out_x =0;
        int min_out_y =0;
        int row =0 ;
        int column =0 ;
        int neighborhood = 0;
        int total_iterations =0;
        double c_delta =0.0;
        double c_alpha =0.0;
        double c_decay=0.0;
        double c_neighborhood =0.0;
        int start_num =0;
        int input_node =0;
        int input_da[1520][2];
        double input_db[1520][8];

/*
This determines the number of passes for the training
*/
        outer_loop_control =layers[Output_X]*layers[Output_Y] * total_passes;

        total_iterations =outer_loop_control * data_samples ;

        c_alpha = alpha;
        c_delta = delta;
        t_passes = total_passes;
        c_often = how_often;
       start_pass = passes;
        c_decay = decay;
        c_neighborhood = control_neighborhood;



/*
This is the variation where the data was all read in in the beginning
In this case all 1520 data sets and 8 features
*/

       for(i=0; i<1520;i++)
       {
        fscanf(fd_data, "%d %d ", &input_da[i][0], &input_da[i][1]);

         for(j=0;j<8;j++)

          {
           fscanf(fd_data, "%le", &input_db[i][j]);
          }
        }
```

```
/*
********************* TESTING ***************************/
*/
            fprintf(stderr,"Start testing  %d\n",pass_num);

             for(vowel_type=0;vowel_type<Vowel_type_max;vowel_type++)
              {
               for(row =Zero;row <layers[Output_X];row++)
                {
                 for(column=Zero;column<layers[Output_Y];column++)
                  {

                     table[vowel_type][row][column] = 0;
                   }
                }
               }


            for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
             {
                    s_num = input_da[vowel_num][0];
                    vowel_class = input_da[vowel_num][1];


                    for(input_node=Zero;input_node<layers[Input];input_node++)
                    {
                     state[input_node] =input_db[vowel_num][input_node +1];
                    }

                    calculate_distance( layers,&min_out_x,&min_out_y);
                    table[vowel_class][min_out_x][min_out_y]++;
             }
/*
At this stage the table array has the node activations storeds in it
Each of the ten layers in table contains the activations for that layer.
The number in each of the elements indicates the number of times that node
was activated by that vowel (indicated by the vowel_class) for that pass
*/

      /************************ CREATE NODE NAMES ****************/


          for(row =Zero;row <layers[Output_X];row++)
           {
            for(column=Zero;column<layers[Output_Y];column++)
             {

                    max_times =0;
                    max_vowel = -1;

             for(vowel_type=0;vowel_type<Vowel_type_max;vowel_type++)
              {
```

```
                    if(table[vowel_type][row][column] >max_times)
                {
                    max_times = table[vowel_type][row][column];
                    max_vowel = vowel_type;
                }
                  }
                   if(max_vowel == -1)
                      printf("\n No activation \n");

                   node_names[row][column] = max_vowel;

                  }
          }

        printf("\n done naming nodes \n");

/*
To create node names choose the winner among tye ten layers for each of the
nodes in the grid. A tie is broken arbitrarily..Here its is the last one that
is the winner
*/


       /**************** CONFUSION MATRIX STUFF ********************/


            for(vowel_num =Zero; vowel_num < data_samples; vowel_num++)
             {

                    s_num = input_da[vowel_num][0];
                    vowel_class = input_da[vowel_num][1];


                    for(input_node=Zero;input_node<layers[Input];input_node++)
                    {
                      state[input_node] =input_db[vowel_num][input_node +1];
                    }


                    calculate_distance(layers,&min_out_x,&min_out_y);

/*
Check to see if the node that was the best for the input has the same index
stored in the node names array as its vowel class indicated by the vowel_class
variable
*/

              if(node_names[min_out_x][min_out_y] != vowel_class )
                 error++;


/*
Increment the confusion matrix count
```

```
*/
                confusion[vowel_class][ node_names[min_out_x][min_out_y]]++;
                }
        printf("\n done creating confusion \n");


        /************************DONE  CREATE NODE NAMES ***************/

/*
Call the confusion function to print the confusion matrix
Call the map_to_table function to print the tables
*/
            map_to_table(&pass_num,&c_often,layers);
            confus(layers, error, data_samples,pass_num );


}
  /* end of control  for testing */
```

```
# Imakefile for the GUI
# Use the imake command as explained in the thesis
#
#

#include <XView.tmpl>

/**/#########################################################
/**/# @(#)Imakefile 1.7 90/08/02 SMI
/**/# Imakefile for examples/canvas

#define InstallSrcs YES

DEFINES = -DSTANDALONE
HEADER_DEST = /home/stu2/g3/jdu4855/X_stuff/lib/xview2/build/include
SYS_LIBRARIES = -lxview -lolgx -lX11 -lm

INCLUDES = -I. -I$(HEADER_DEST) -I$(TOP)
DEPLIBS = XViewClientDepLibs
LOCAL_LIBRARIES = XViewClientLibs

SRCS =  kohonen_map.c  graph_kohonen.c init.c memory.c read.c memory.c\
        calc.c my_tab.c adjust.c write.c confuse.c

INSTALL_SRCS = $(SRCS)
ALLFILES =  $(SRCS)

PROGRAMS=kohonen kohonen_map

AllTarget($(PROGRAMS))

SingleProgramTarget(kohonen_map, kohonen_map.o, $(LOCAL_LIBRARIES), /**/)

SingleProgramTarget(kohonen, graph_kohonen.o init.o memory.o read.o calc.o \
        my_tab.o adjust.o write.o confuse.o , $(LOCAL_LIBRARIES), /**/)

graph_kohonen.o :
        cc graph_kohonen.c -c -g  \
        -I/home/stu2/g3/jdu4855/X_stuff/lib/xview2/build/include \
        -I/usr/include -I.  $(CFLAGS) -o graph_kohonen.o

init.o:
        cc -target sun3 -c -g init.c -o init.o
memory.o:
        cc -target sun3 -c -g memory.c -o memory.o
calc.o:
        cc -target sun3 -c -g calc.c -o calc.o
read.o:
        cc -target sun3 -c -g  read.c -o read.o
my_tab.o:
        cc -target sun3 -c -g  my_tab.c -o my_tab.o
write.o:
        cc -target sun3 -c -g  write.c -o write.o
```

```
confuse.o:
        cc -target sun3 -c -g  confuse.c -o confuse.o
adjust.o:
        cc -target sun3 -c -g  adjust.c -o adjust.o
```

```
#This is the makefile for use with the GUI for running the net
#This can be used as a substitute if you are not able to use imake

Exec = kohonen

Lib = -lxview -lolgx -lX11 -lm

CFlags = -g


Objects = init.o \
        memory.o \
        read.o \
        calc.o \
        my_tab.o\
        adjust.o\
        write.o\
        confuse.c

$(Exec) : graph_kohonen.o $(Objects)
        cc graph_kohonen.o $(CFLAGS) $(Objects)  $(Lib) -o $(Exec)

graph_kohonen.o :
        cc graph_kohonen.c -c -g \
        -I/home/stu2/g3/jdu4855/X_stuff/lib/xview2/build/include \
        -I/usr/include -I.  $(CFLAGS) -o graph_kohonen.o

init.o:
        cc -target sun3 -c -g init.c -o init.o
memory.o:
        cc -target sun3 -c -g memory.c -o memory.o
calc.o:
        cc -target sun3 -c -g calc.c -o calc.o
read.o:
        cc -target sun3 -c -g  read.c -o read.o
my_tab.o:
        cc -target sun3 -c -g  my_tab.c -o my_tab.o
write.o:
        cc -target sun3 -c -g  write.c -o write.o
confuse.o:
        cc -target sun3 -c -g  confuse.c -o confuse.o
adjust.o:
        cc -target sun3 -c -g  adjust.c -o adjust.o
```

```
#       This file has the command to start the make for the three
#       systems. Look in page E-5 for makefile.sunos


uSun    :
        /bin/make -f makefile.sunos

uUlt    :
        /bin/make -f makefile.ultrix

u3b2    :
        /bin/make -f makefile.sysv
```

```
# makefile.sunos
#
# The executable
Exec = kohonen
#The library that is needed
Lib = -lm
#The cc flags..use -O for non-debugging purposes
CFlags = -g

#The object files

Objects = main.o \
        init.o \
        memory.o \
        read.o \
        control.o \
        calc.o \
        adjust_sun.o  \
        write.o \
        confuse.o \
        data.o \
        my_tab.o

# adjust_sun.c is just a variation of adjust.c that has been modofied to run
#on the suns
#
#
#here is where it all happens

$(Exec) : $(Objects)
        cc $(CFLAGS) $(Objects)  $(Lib) -o $(Exec)
```