Harmonic Analysis Using Neural Networks

by

Wan Shun Vincent Tsui

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

©Copyright by Wan Shun Vincent Tsui 2002

Harmonic Analysis Using Neural Networks

Master of Applied Science, 2002

Wan Shun Vincent Tsui

Edward S. Rogers Sr. Department of Electrical and Computer Engineering

University of Toronto

Abstract

This thesis describes a system of neural networks that performs harmonic analysis on musical compositions. The system is split into four groups of Networks: Key Network, Root Network A (used to detect secondary functions), Root Network B, and Quality Network. The system is trained on 20 J.S. Bach chorales and tested on another 18 Bach chorales. Accuracies of over 90% were obtained for all four groups of Networks. Comparison with a modified version of Krumhansl's key-finding algorithm indicates Key Network outperforms it by more than 16% in overall accuracy. This work has potential for extension into music of other composers or genres. Also, this work can be used by a computer accompanist to determine the key of an improvising soloist.

Acknowledgement

First of all, I wish to thank gratefully Prof. James MacLean for supervising me for the entirety of this thesis. His suggestions have impacted this work very much.

I also thank my fellow graduate schoolmates in the SF lab for making me feel I am not alone in this seemingly neverending world of graduate studies.

I would like to thank University of Toronto for all the forms of financial support it had provided me.

I would like to thank my brothers and sisters in Paul Fellowship for praying for me, my cousin Alan Wong for his continual encouragement to go on, my parents and grandmother for encouraging me to pursue graduate school, and Brenda Chan for keeping me sane in the days and nights when work is much but progress is few.

Finally, I give thanks to my Lord Jesus Christ for leading me through this earthly travail. Without Him, nothing is possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

In music, harmonic analysis is done by a music theorist on a piece of music to gain an in-depth understanding of it. Music can be appreciated in many ways; harmonic analysis is one of the more technical ways. It helps the analyst to understanding the tools and techniques that the composer used in writing that piece of music. Music, at least so-called classical music, is very structured. There are a lot of rules that a composer needs to follow. However, one thing that sets a great composer apart from mediocre ones is that a great composer knows how and when to bend intelligently these rules. Normally, when musical rules are broken, a trained ear reacts with displeasure. However, when this is done by a great composer, who has superb skills, a trained ear reacts with joy and amazement. Harmonic analysis is a way for a trained eye to react the same way as a trained ear would.

Harmonic analysis is not an easy task. It requires a solid background in musical rudiments. It is natural to ask the question: is there an algorithm to perform harmonic analysis? A search through literature revealed only

two serious attempts at harmonic analysis ([22], [17]). Both use essentially rule-based techniques. The general disadvantage of rule-based techniques in solving a complex problem such as harmonic analysis is that the solution space is big, and it is hard to devise a finite set of rules to cover it properly.

This thesis proposes a neural network solution to the harmonic analysis problem. Neural networks are good at solving problems which have no apparent and easy rule-based solutions. In effect, a neural network learns and internalizes the rules when it is repeatedly shown many instances of the problem and the corresponding solutions. In other words, a neural network learns by example. The input to the system of this thesis is the music; the output is the harmonic analysis. The system will be repeatedly shown different instances of music and their corresponding harmonic analyses. In the end, it is hope that when the system is shown a new instance of music, it will have learned well enough to produce the correct harmonic analysis.

Chapter 2

Background

This Chapter is divided as follows. Section 2.1 gives a background on certain musical concepts, starting from basic definitions and working towards the concept of harmonic analysis. Section 2.2 gives a background on neural networks. Section 2.3 is a literature review on relevant topics.

2.1 Overview of Certain Musical Concepts

In order to understand and appreciate harmonic analysis fully, it is imperative to have a thorough background in music theory. This section will provide some of this background, enough for the reader to understand the task at hand. For a more complete treatment, or for information on how to read music, please consult a music theory textbook, such as [12].

2.1.1 Musical Styles

Music exists in practically all cultures of the world. Amongst other things, it is a way to express one's emotions. Different cultures have developed their own styles of music. Thus, Chinese music is different from Indian music, which is different from African music. Although some musical concepts discussed in this Section apply to all musical styles, the general style of music referred to in this thesis is Western music, which can be said to be the musical idiom that existed in Europe from 1650 to 1900. Such famous composers as Johann Sebastian Bach (1685-1750), Wolfgang Amadeus Mozart (1756-1791), and Ludwig van Beethoven (1770-1827) all belonged to this style.

2.1.2 Definitions of Basic Musical Terms

Any music consists of two main components: *pitch* and *rhythm*. *Pitch* is the frequency of a sound. *Rhythm* is the durations of different pitches. Thus, in the plainest terms, a piece of music consists of a sequence of combinations of sounds at different frequencies for different durations. Two pitches are said to be an *octave* apart if the frequency of one pitch is twice that of the other. To the human ear, two pitches an octave apart sound very similar [16]. An interval is the difference in frequency between two pitches. In Western music, the octave interval, or plainly the octave, is divided into twelve equal intervals. Pitches are labeled using the first seven letters of the Western alphabet. Their locations on a keyboard are shown in Figure 2.1.

Pitches that differ by multiples of an octave are given the same label. By convention, the first A pitch above middle C is known as "concert A" and is

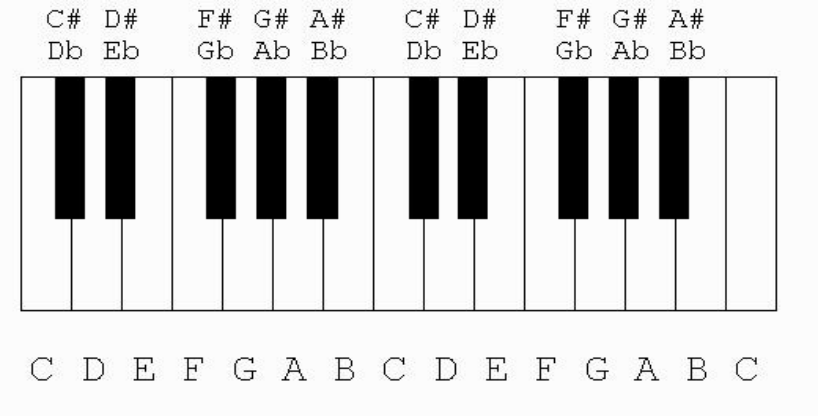

Figure 2.1: Keyboard with letter names shown.

set at 440Hz. (Middle C is the C that is nearest to the center of the piano.) The rest of the pitches are set such that two notes that are one semitone apart (e.g. A and A♯) have a frequency ratio (e.g. freq(A♯)/freq(A)) of $2^{1/12}$.

In almost all pieces of Western music, not all of the twelve pitches within an octave appear equally frequently. A *mode* is a particular subset of these twelve pitches. An example of a mode is *major*. The major mode consists of seven pitches. The sequence of pitches in a mode is called a *scale*. There is only one sequence per mode. A scale is labeled by its first pitch. Define a *semitone* to be the interval between two neighbouring pitches. Thus, C and C♯ are a semitone apart, as are E and F. Define a *wholetone* to be twice the interval of a semitone. Given the first pitch X_1, the X_1 *major scale* is the sequence $\{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$, with I_{12} = whole tone, I_{23} = whole tone, I_{34} = semitone, I_{45} = whole tone, I_{56} = whole tone, I_{67} = whole tone, where I_{ab} is the interval between pitches X_a and X_b, X_b being the pitch with the higher frequency. This definition is illustrated in Figure 2.2. As an example, the C major scale is the sequence {C, D, E, F, G, A, B}. Looking

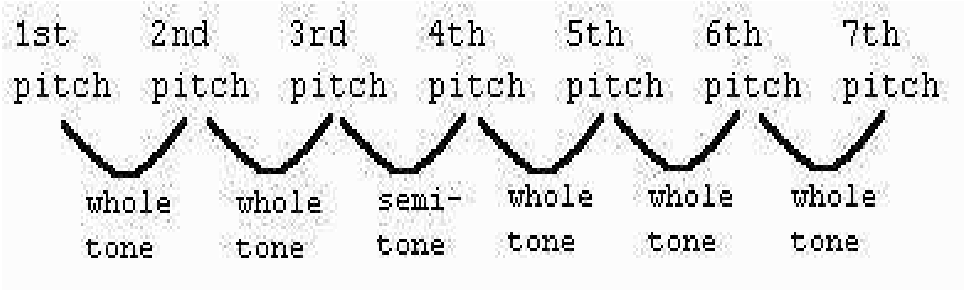

Figure 2.2: Illustration of the relationships between the seven pitches of a major scale. A semitone is the frequency difference between two neighbouring pitches. A whole tone is two semitones. The pitches are ordered from lowest to highest frequencies.

at the keyboard, these are all the white keys. Another mode is the *minor* mode. There are three variations to the minor mode; only one of them, the natural minor, will be looked at here. Given the first pitch X_1, the X_1 *natural minor scale* is the sequence $\{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$, with I_{12} = whole tone, I_{23} = semitone, I_{34} = whole tone, I_{45} = whole tone, I_{56} = semitone, I_{67} = whole tone, where I_{ab} is the interval between pitches X_a and X_b, X_b being the pitch with the higher frequency. For example, the A natural minor scale is the sequence {A, B, C, D, E, F, G}. While the C major and the A natural minor scales contain the same pitches, the difference between the two is that the ordering of the notes is different. Note that scales can start on any of the twelve pitches. Given the starting pitch, the rest of the scale can be constructed using one of the above definitions. One rule about naming the pitches in a particular scale is that the letter names must be used in succession (the letter following G is A). Thus, the D major scale is the sequence {D, E, F♯, G, A, B, C♯} and cannot be {D, E, G♭, G, A, B,

C♯} nor {D, E, F♯, G, A, B, D♭}, even though the letter names refer to the same sequence of pitches. Two letter names that refer to the same pitch are said to be *enharmonically equivalent*. Thus, F♯ and G♭ are enharmonically equivalent.

A *note* is the sounding of a particular pitch for a particular duration. A *chord* is the simultaneous sounding of two or more notes. A *sonority* is "the set of all sounding notes that are present whenever any note is articulated." [20, page 22] The difference between chords and sonorities is that chords are usually taken to mean some pre-defined combination of notes (e.g. those discussed in Section 2.1.11), while sonorities can be any arbitrary combination of notes.

2.1.3 Key

The concept of *key* is loosely defined as follows: a piece of music is said to belong to key X if pitch X is most prominent in this piece of music. Pitch X is prominent due to two reasons: 1) the pitch X appears frequently; 2) other pitches which have a close relationship with pitch X appear more often than those pitches which do not. The concepts of key and mode are greatly intertwined. A piece of music is said to belong to X major (minor) if the pitch X appears frequently, and the (other) pitches of the X major (minor) scale appear more often than other pitches. Another way to view this is to say that the notes of a piece of music in key X "revolves around" or "pulls toward" pitch X. For example, the famous Moonlight Sonata by Beethoven belongs to C♯ minor. If one looks at its musical score, one finds that the note C♯ appears frequently, and the (other) notes of the C♯ minor scale appear

more than other notes not belonging to this scale. If one were to listen to this music and had perfect pitch (the ability to recognize the letter names of pitches by ear), one would hear C♯ featured prominently throughout.

2.1.4 Modulation

The fact that a piece of music belongs to a certain key does not preclude it from visiting other keys somewhere in the middle. *Modulation* is the change of key within a piece of music. This means that within a certain passage in the piece, the most prominent pitch changes from the one the piece started with to another one. For example, while the overall key of the Moonlight Sonata is C♯ minor, there is a passage within it where modulation occurs, and the new key is B minor. Modulation can occur many times throughout a composition. In the Moonlight Sonata, in addition to the aforementioned modulation, the key changes several more times. However, in almost all pieces of Western music (and in the Moonlight Sonata as well), no matter what keys the piece has modulated to, the key it ends on is the same key it started with.

2.1.5 Tonicization

Simply put, *tonicization* is a "mini-modulation." The musical passage temporarily visits a foreign key but does not stay long enough there for a listener to perceive a modulation. The word tonicization means the *tonic chord* (defined below in Section 2.1.8) of a certain foreign key is emphasized. The difference between modulation and tonicization is quite subjective [12]. The

most important difference is time. The longer a passage stays in the foreign key, the more likely a listener will perceive it as having modulated.

2.1.6 Transposition

Transposition is the manual shifting of an entire piece of music from one key to another. A piece of music is written in a certain key, but it can be transposed into any arbitrary key. The relative positions of the notes in the piece are maintained when it is transposed. Other than being an academic exercise, transposition is useful because a piece may be easy to play on instrument A in key X, but, in order for it to be easy to play on instrument B, it needs to be transposed into key Y. Moreover, for the same instrument (e.g. piano), a piece may be easier to play in key X rather than key Y, so the composer may opt for one or the other depending on whether that added difficulty is desired.

2.1.7 Triads

Define the nth degree of scale X as the nth term in its sequence. For example, the first degree of the C major scale is C, while the fifth degree of the same scale is G. Given n, the triad built upon the nth degree of scale X is the chord consisting of the three notes $\{X_n, X_{(n+2)mod7}, X_{(n+4)mod7}\}$. For example, the triad built upon the first degree of the C major scale is the chord consisting of the three notes {C, E, G}; the triad built upon the fifth degree of the same scale is the chord {G, B, D}. Triads are used extensively in Western music because when played, they give a harmonious sound.

Degree Ordinal	Degree Name
1st	tonic
2nd	supertonic
3rd	mediant
4th	subdominant
5th	dominant
6th	submediant
7th	leading note

Table 2.1: The degree names for each of the seven notes of both the major and minor scales.

2.1.8 Names for the degrees of the scale

There are seven degrees to both the major and minor scales. Each degree is given a name. The first degree is called tonic; second, supertonic; third, mediant; fourth, subdominant; fifth, dominant; sixth, submediant; seventh, leading note. This is also presented in Table 2.1. The meaning of each name is beyond the scope of this overview. Thus, instead of saying "the triad built upon the first degree of the C major scale is the chord {C, E, G}," it is equivalent to say "the tonic chord of C major is {C, E, G}." Each of the seven triads is notated with the Roman numeral corresponding to the scale degree the triad is associated with. Hence, the above phrase can also be expressed as "the I chord of C major is {C, E, G}." The different triads within a given key serve distinct musical functions. It is these functions that are interesting to analyze.

2.1.9 Secondary Functions

In addition to the I...VII chords defined above, there are also chords involved in tonicization that need labeling. Tonicization usually involves two chords: 1) the tonic chord of a foreign key; 2) the dominant chord of the same foreign key, which immediately precedes the tonic chord of the foreign key. Now, the tonic chord of the foreign key can always be analyzed in the current key. For example, imagine that there is a chord with notes {C, E, G}, and the current key is A minor. Then, this chord is the tonic chord of the foreign key of C major, while in the current key of A minor, it would be the mediant, or III, chord. The chord preceding this would be the dominant, or V, chord of the foreign key of C major; hence, in the current key of A minor, this chord is notated as the dominant chord of the mediant chord, or, more briefly, a V of III or a V/III chord. This type of chord is called the *secondary function* because, in the context of the current key, these chords serve a function of another key. This is a device commonly used by composers to make a passage sound more interesting by giving hints of another key but not actually modulating there.

2.1.10 Functional Analysis

Consider any piece of music. It contains a certain number of sonorities. Each sonority belongs to a certain key (except at points of modulation, where it can belong to two keys at once). Loosely, given the key of each sonority of a piece of music, *functional analysis* is the classification of each sonority into one of the seven different triads of the given key of that sonority. Strictly speaking,

however, in certain situations, certain degrees of the scale (especially minor scale) are raised (i.e. one semitone higher than normal) or lowered (one semitone lower than normal), and triads can be built upon them, so there are more than seven triads possible. Also, an extra note can be placed on top of a triad, making it a four-note chord, and functional analysis must include these chords as well.

2.1.11 Qualities of Chords

The *quality* of a chord is the set of intervals between the notes of the chord. In Western music, nine chords with different qualities commonly occur. They are listed in Table 2.2.

It is understood that the notes of the chord are ordered from lowest to highest frequency. The *root* of the chord is defined to be its 1st note. A chord can be distinguished by its root and quality. Thus, the C major chord is {C, E, G}; the E dominant 7th chord is {E, G♯, B, D}.

2.1.12 Harmonic Analysis

Essentially, *harmonic analysis* is 1) the determination of key of each sonority, and 2) functional analysis. Neither of these is an easy problem to solve. The first is difficult because one cannot just arbitrarily select a passage, count which note occurs most often, and doggedly claim that all sonorities in this passage belong to that key. There is some literature that describe algorithms for finding keys for a piece of music (more on this in Section 2.3). The second is difficult because in some sonorities, their notes do not correspond to any of the common chords. The common chords consist of at least three notes, but

Quality	Interval from 1st to 2nd note (semitone)	Interval from 2nd to 3rd note (semitone)	Interval from 3rd to 4th note (semitone)
Major	4	3	N/A
Minor	3	4	N/A
Diminished	3	3	N/A
Augmented	4	4	N/A
Dominant 7th	4	3	3
Major 7th	4	3	4
Minor 7th	3	4	3
Diminished 7th	3	3	3
Half-diminished 7th	3	3	4

Table 2.2: Commonly occurring qualities of chords. The quality of a chord is the set of intervals between the notes of the chord. Other qualities do exist, but they do not have names.

a sonority may contain two, or four, or even five. Conversely, a sonority may contain all the notes of a particular common chord, but depending on the metrical importance (explained in Section 2.1.15) of this sonority relative to the sonorities surrounding it, as well as on the notes of the sonorities surrounding it, it may not be classified as that particular common chord.

The harmonic analysis described up to this point is what this thesis attempts to perform. Harmonic analysis can be taken further to include other information. One such piece of information is the classification of non-chord tones. A *non-chord tone* is a note of a sonority that does not belong to that sonority's chord classification. For example, if a sonority has notes {C, E, F, G} and is classified as a I chord in the key of C major, then the note F is a non-chord tone. There are many classes of non-chord tones. A non-chord tone can generally be classified according to the relative positions of the notes in the same voice, or part (defined below in Section 2.1.13), in its previous and subsequent sonority. Another piece of information that can be part of harmonic analysis is inversion analysis. Inversion analysis determines which note of the chord is positioned lowest in the musical score. These two tasks are relatively straightforward and hence omitted from the harmonic analysis of this thesis.

2.1.13 Four-Part Chorales

Chorales are Protestant hymns that came into being during the Reformation [16]. A leading figure in the Reformation, Martin Luther, wrote many chorales. By itself, musically speaking, a chorale only consisted of a melody tune. In order to make chorales sound richer harmonically, later composers,

notably Johann Sebastian Bach, undertook the task of harmonizing some of these chorales. Harmonizing a chorale tune means adding several, typically three, parallel tunes to the original tune, so that several voices, typically two voices of women and two voices of men, can each sing a different tune (but with the same words), yet together would sound harmonious and pleasing to the ear. The new tunes added would typically be lower than the original, so that the original, being highest in pitch, would still be heard clearly. Almost all of the chorales that Bach harmonized, and all that were trained and tested in this thesis, consist of four voices, or parts. The higher voice for women is called *soprano*; the lower voice for women is called *alto*; the higher voice for men is called *tenor*; the lower voice for men is called *bass*. Each voice has his/her own range in pitch, and, while the different ranges do overlap one another, for any given sonority, the order of voices appearing in it, from highest to lowest, is usually soprano-alto-tenor-bass. A part of a Bach chorale, together with its harmonic analysis, is given in Figure 2.3.

2.1.14 Application of Harmonic Analysis

Performing harmonic analysis on a piece of music will uncover many musical techniques that the composer used to make the piece sound the way it does. Many musical effects can be dissected using harmonic analysis. For example, upon hearing a passage of music, one detects that modulation has occurred; however, it may be difficult to pinpoint where or how the modulation occurred. By going to the score and performing harmonic analysis on it, one may see that the composer used a special type of chord, and it is at this point that the passage has, very smoothly and subtly, gone to the new

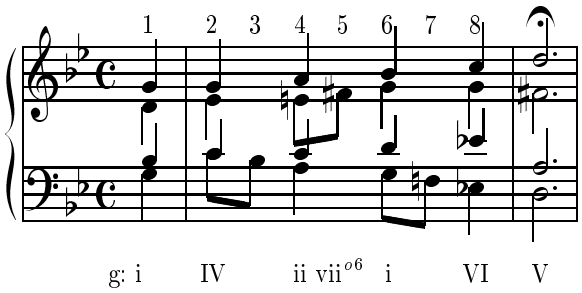

Figure 2.3: A passage from Bach chorale #28 of the Kalmus [4] edition with harmonic analysis included. The small letter 'g' indicates that this passage is in the key of G minor (G major would be denoted 'G'). The Roman numerals embed some of the quality information. Capital Roman numerals generally indicate major chords and small Roman numerals generally indicate minor chords, except when other symbols are added. For example, the 'o' to the right and above of 'vii' indicates this to be a diminished 7th chord. The '6' is inversion information (not relevant to the purposes of this thesis). The numbers above the score are the sonority numbers and are not part of the score nor the harmonic analysis.

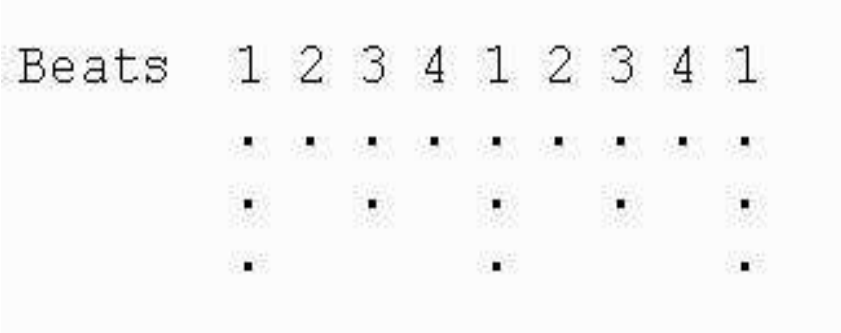

Figure 2.4: Hierarchy of beats in a 4/4 meter. More dots means higher in the hierarchy.

key. Being able to perform harmonic analysis offers another way of appreciating a piece of music. It also offers a way of comparing compositions by the same composer, or even different composers. For example, one can compare the frequency of usage of different chords in compositions by Bach versus Beethoven.

Goldman [8] has given an excellent reason why learning harmonic analysis is relevant today. He wrote, "The student today is in all probability not going to compose in the idiom of the Bach chorales... In some respects, he no longer needs traditional or classical harmony as a *technique*. What he does need is an understanding of, and a feeling for, the harmonic principles that form the basis of his artistic heritage."

2.1.15 Representation of Metrical Structure

When hearing a piece of music, the listener detects the regular occurrence of musical events. These events are called beats, and they are marked by such physical actions as nodding heads, clapping hands, or tapping feet. Not all beats are equal; some are stronger (i.e. given more emphasis) while others

are weaker (i.e. given less emphasis). These patterns of strong and weak beats are called meter [15]. For example, in a 4/4 meter, it is commonly understood that the strongest beat in the bar is the first beat, followed by the third beat, followed by the second and fourth beats. Here it is seen that there are three levels of beats. To represent these beats, in Lerdahl's book [15], a number of dots are placed at the occurrence of each beat. The beats at the lowest level are represented by one dot; the beats at subsequent higher levels are represented by one more dot than the beats at the previous lower level. The hierarchy of beats in a 4/4 meter is thus represented as in Figure 2.4.

2.2 Overview of Neural Networks

2.2.1 Their Structure

A neural network is "a computing system made up of a number of simple, highly interconnected processing elements, which processes information by its dynamic state response to external inputs." [7, page 47] Unlike a conventional computer, which has separate units of CPU and memory, a neural network processes and stores everything at its many nodes. The word "neural" in neural network comes from the fact that the structure of a neural network is in some ways similar to that of the brain. The brain consists of many neurons; the neurons communicate with each other via electrical pulses. Since storage in a neural network is decentralized, no single neuron would contain critically important information. If a hard drive had a bad sector, all data in that sector would be inaccessible. In contrast, if some neurons in a neural network were

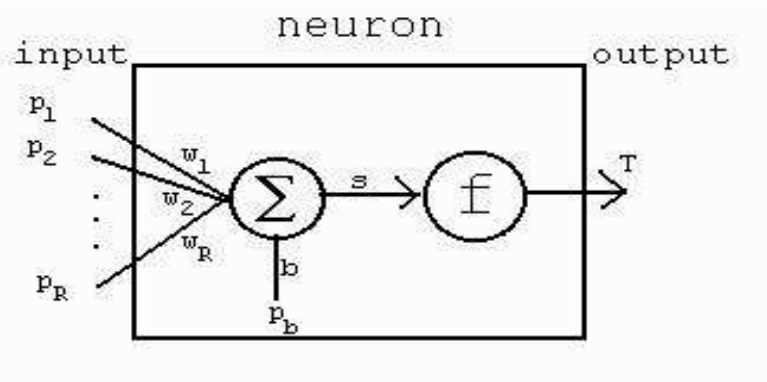

Figure 2.5: Structure of a neuron. A neuron takes as input $p_1, ..., p_R$. Each input p_i has an associated weight w_i. A neuron may also have a bias input p_b and an associated weight w_b. The sum is computed as $s = \sum_{i=1}^{R} w_i p_i + w_b p_b$. The sum goes through a transfer function f, and the output of the neuron is computed as $T = f(s)$. T may go to one or more other neurons in the network.

turned off, then the overall performance would drop, but no specific area would be severely affected.

In any neural network, there are three types of neurons: those that receive information from the outside world are called input units; those that provide information to the outside world are called output units; those that both receive and provide information within the neural network are aptly labeled hidden units.

The structure of a single neuron is as follows [3]. A neuron takes in a certain number, R, of inputs, each a real number. Label these as p_1, ..., p_R. Associated with each input p_i is a weight w_i, also a real number. Optionally, a neuron may also have another input, called a bias input. Label it as p_b. The bias input also has an associated weight w_b. The value of p_b is fixed.

The sum s is computed of all the weighted inputs (including bias):

$$s = \sum_{i=1}^{R} w_i p_i + w_b p_b. \tag{2.1}$$

s is not the final output of the neuron. The final output T is obtained by taking s through a transfer function f:

$$T = f(s). \tag{2.2}$$

Depending on the type of network, f is chosen differently. Unless the neuron is an output neuron, the output of the neuron becomes the input of one or more other neurons in the network. This is illustrated in Figure 2.5. When all the neurons in the neural network work together, mathematically speaking, it computes a function F, F: $R_m \mapsto R_n$, where m is the number of input neurons, and n is the number of output neurons.

2.2.2 Network Training

In order for a network to be useful, it needs to be trained. Training is the process of updating a network's weights, so that the function F it computes is as close as possible to some desired function F'. The network usually starts off with random weights; there exist algorithms that iteratively update the weights when training data is repeatedly presented to the network. One such common algorithm, known as gradient descent [6], is outlined below. The idea is that the network attempts to learn the underlying F' given its sample points represented by the training data. If the network has trained well, when given completely new testing data, it would closely imitate the behaviour of F'.

There are two types of training: supervised and unsupervised. Supervised training is what is described above: the training data includes both the input and the corresponding expected output. In unsupervised training, only input data is provided, and the network infers pattern from them directly. The problem of harmonic analysis is, by nature, suited to supervised training. The training data includes the expected output of key, mode, and the like, of each sonority. Hence, networks that perform unsupervised training will not be further discussed.

2.2.2.1 Gradient Descent

Essentially, gradient descent updates the weights of the network in the direction of the negative gradient of the error function. The error function of the network is a measure of how much the current actual outputs of the network are away from the training outputs. At the beginning of each iteration, the error function is computed. Then, its gradient is determined (typically by the error backpropagation algorithm, outlined in Section 2.2.2.3). Finally, the weights are updated by a scale-factor of the negative gradient. This scale-factor is known as the learning rate. Fixing the learning rate has two problems: 1) it is difficult to tell beforehand a good learning rate for the particular network at hand; 2) a different learning rate may be suitable for different areas of the error surface. Hence, an improvement over the standard gradient descent is to allow the learning rate to vary. A further improvement is to add a momentum term. This takes the changes to the weights in the previous iteration into account. This has the advantage of being able to carry the descent over small "valleys," or dips in the error surface.

2.2.2.2 Scaled Conjugate Gradient

The scaled conjugate gradient [6] algorithm is a training algorithm that has been shown to be more powerful than gradient descent. Scaled conjugate gradient is based upon conventional conjugate gradient, which in turn is one algorithm in the class of parameter optimization strategies known as line search. Iteratively updating the weights of a network is the same thing as taking a sequence of steps through a weight space. Two parameters are involved when each step is taken: 1) the direction; 2) the distance. In gradient descent, the distance taken is determined by the learning rate parameter and the magnitude of the gradient (and, possibly additionally, the momentum). In line search, the distance taken along a given direction is determined by minimizing the error function along the given direction. In conventional conjugate gradient, the distance taken is done in this manner. The direction is taken such that the component of the direction's gradient parallel to the previous direction must remain zero (to lowest order). The scaled conjugate gradient algorithm improves upon conventional conjugate gradient by avoiding the line-search procedure, and hence can sometimes offer a significant improvement in speed [6].

2.2.2.3 Error Backpropagation

The goal of error backpropagation is to determine the derivative of the error function with respect to each and every weight in the network. First, for unit j, define

$$\delta_j \equiv \frac{\partial E^n}{\partial s_j}, \tag{2.3}$$

where E^n is the error function for training pattern n, and s_j as defined in equation 2.1. The backpropagation formula is

$$\delta_j = f'(s_j) \sum_k w_{kj} \delta_k, \tag{2.4}$$

where $f'()$ is the derivative of the transfer function for unit j, w_{kj} is the weight for the connection to unit k from unit j. Since δ_j is dependent on a weighted sum of the δ_k's, which are from the units δ_j outputs to, the errors are propagated backwards through the network, hence the name of the algorithm. As can be seen, this is a recursive formula. The base case is an output unit k:

$$\delta_k \equiv \frac{\partial E^n}{\partial s_k} = f'(s_k) \frac{\partial E^n}{\partial y_k} \tag{2.5}$$

where $y_k = f(s_k)$. Finally, the derivative of the error function with respect to each and every weight is as follows:

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j s_i. \tag{2.6}$$

These are the derivatives for one error function. This is repeated for each training pattern, and the derivatives are summed:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E^n}{\partial w_{ji}}, \tag{2.7}$$

where $E = \sum_n E^n$ is the error function for all training patterns. This is exactly the gradient in weight-space needed in the training algorithms described above in Sections 2.2.2.1 and 2.2.2.2.

2.2.3 Applications of Neural Networks

Neural networks are designed to solve problems that conventional, rule-based algorithms have a hard time solving. As its name implies, a rule-based algo-

rithm would not solve very well a problem that is very hard to break down into lists of rules. Many problems in real-life fall into this category because 1) one does not know where to begin, and 2) even if one came up with a set of rules, there would almost always be exceptions or special cases that were missed. One broad class is pattern recognition and classification. How is handwriting recognized? How is a face recognized as being a familiar one, a look-alike of a familiar one, or an altogether unfamiliar one? It is not immediately apparent how one might design a rule-based algorithm to solve these problems. However, when neural networks are trained on known instances of handwriting or faces, they would be able to recognize distinctive features of different letters or faces, store these features in the weights, and predict to a high degree of accuracy new instances of letters or faces.

The problem of harmonic analysis is a problem of pattern classification. For example, out of many possible keys a sonority may belong to, the network picks the one most likely to be correct.

2.2.4 Different Kinds of Neural Networks

The kind of network most commonly used is called a feed-forward network. In a feed-forward network, the placement of neurons and their interconnections are completely arbitrary, with one exception: there must be no feedback loops. This means that if successive numbers were attached to the input, hidden, and output units, each unit (except input) would only receive connections from units with a smaller label [6]. A special class of feed-forward networks is layered feed-forward networks. Such networks contain zero or more layers of hidden units, in addition to the input layer of units and the

output layer of units. There are interconnections between every unit in one layer and every unit in the next layer; otherwise, there are no other interconnections.

Two types of transfer functions are generally used in a feed-forward network: linear and logistic sigmoid. Linear transfer functions have the form $f(s) = as + b$. Logistic sigmoid transfer functions have the form $f(s) = 1/(1 + \exp(-s))$. (The hyperbolic tangent ('tanh') function is also commonly used; it is actually equivalent to the logistic sigmoid if two linear transformations were applied, once at the input and once at the output.) Sigmoidal transfer functions are important in the following way. A feed-forward network with one layer of hidden units, if the transfer functions of the hidden units are sigmoidal, "can approximate arbitrarily well any functional (one-one or many-one) continuous mapping from one finite-dimensional space to another, provided the number M of hidden units is sufficiently large." [6, page 130] A corollary of this property is that this kind of network, if dealing with a problem of pattern classification, can approximate any decision boundary to any arbitrary accuracy. A decision boundary is simply a boundary in multi-dimensional space that separates one class of inputs from another. In feed-forward networks with one or more hidden layers, if the transfer function used in the output units is the logistic sigmoid, and if the class-conditional densities belong to the family of exponential distributions, then the output of the network can be considered as posterior probabilities. This provides a theoretical background for the choice of the networks used in this thesis to perform harmonic analysis. Most of the networks used in this thesis are feed-forward networks with one hidden layer; given a large number of hid-

den units, the networks can approximate the decision boundary to a high degree; the output of the networks can be taken to mean the probability of this sonority being in a certain key, for example.

A second type of neural networks commonly in use is the radial basis network. In a radial basis network, the interconnections are feed-forward in nature, and typically there is only one layer of hidden units. However, the transfer functions of the hidden units are different from those in the feed-forward networks. In a radial basis network, the transfer function in each hidden unit is termed a basis function; the parameters of the basis functions are determined during training. For example, the Gaussian is commonly used as a basis function:

$$\phi_j(x) = \exp\left(-\frac{\|x - \mu_j\|^2}{2\rho_j^2}\right) \tag{2.8}$$

where, for hidden unit j, x is the vector of inputs from the input layer, μ_j is the vector that determines the center of basis function ϕ_j, and ρ_j is the variance. As can be seen, input close to the center of the basis function will cause a large activation from this hidden unit. Also, different inputs which are the same distance away from the center of the basis function will cause equal activation from the hidden unit.

The outputs of the network are a linear combination of the outputs of the basis functions:

$$y_k(x) = \sum_{j=1}^{M} w_{kj}\phi_j(x) \tag{2.9}$$

where, for output unit k, w_{kj} is the weight corresponding to hidden unit j, and M is the total number of hidden units.

Both the feed-forward and the radial basis neural networks offer similar potential for performing well in the problem of harmonic analysis. However,

due to time constraints, only one was chosen for further investigation. Training a feed-forward network is less complicated than training a radial basis network. Training a radial basis network requires two stages, while training a feed-forward network requires only one. Both types of networks need the number of hidden units specified. However, the radial basis network needs, in addition, the parameters of each basis function specified as well. Also, because of the localized nature of the basis functions, a trained radial basis network can give unstable performance on new test data. These are the reasons feed-forward networks were chosen over radial basis networks.

A third and final type of network to be discussed is an Elman network [3]. An Elman network is very similar to a feed-forward network, except that it contains feedback loops from the output of the hidden units to the input of the hidden units. Because the input contains information from the previous time-step, this type of network is capable of learning temporal, in addition to spatial, patterns. Music very much exists in the temporal domain, and therefore it is thought that using this type of network in the problem of harmonic analysis may be helpful.

2.2.5 Committee of Networks

An often-used technique to improve the prediction accuracy of neural networks is to train different instances of networks and to combine them in the form of a committee [6]. The different instances may be of different network architectures or modeling of the problem. Bishop [6] describes the output of the committee to be an average or weighted average outputs of the members of the committee. For a classification problem, such as the one being dealt

with in this thesis, a further step is to take the largest output unit as the output of the committee. For the case where the output of the committee is simply an arithmetic average of the outputs of the members of the committee, the sum-of-squares error from the intended function f of the output of the committee can be reduced as much as by a factor of N from the average error of the members of the committee, where N is the number of members in the committee. In most cases, however, the error is reduced by much less than a factor of N; this is because the errors of the members are usually highly correlated. Hence, the challenge is to pick networks whose errors are similar in magnitude but are quite uncorrelated.

As shall be explained in the Chapter 4, a slight modification of the committee technique was employed. The modification is as follows. First, pick the largest output of each member network. Then, from these largest outputs, pick the output that occurs the most. For example, if the committee contains three member networks, and the largest output for network 1 and 2 is G major, and the largest output for network 3 is C major, then the output of the committee is G major. If some or all of the largest outputs occur equally often, then the output of the committee is randomly chosen among the largest outputs that occur equally often. For example, if the largest output for network 1 is C major, for network 2 is G major, and for network 3 is A minor, then the output of the committee is randomly chosen among these three outputs. From the trained networks, five sets of committees of 3 were randomly chosen, and it was found that an average of only 0.8% of the outputs fall into this last case. A more sophisticated extension to this scheme would be to pick the network whose largest output is the largest

amongst all member networks of the committee. For example, if the largest output for network 1 is C major and has a value of 0.7, for network 2 is G major and has a value of 0.75, for network 3 is A minor and has a value of 0.8, then the output of the committee is A minor. However, it is not clear whether this extension works better than the original random scheme. Due to time constraints, this extension was not tried, but it can be deduced that the performance gain can be no more than $0.8\% - \frac{0.8\%}{3} = 0.53\%$.

This new form of the committee is reasonable in the context of the present problem. Because of the network architecture, the output of the network can be taken to represent posterior probabilities. And, since this is a pattern classification problem where only one pattern is chosen, the most natural procedure is to select the largest unit, representing the most likely occurring one, among all the output units. Finally, a vote is taken from all networks, and the majority of the votes is taken to be the output of the committee.

2.3 Literature Review

This section presents a review of some related works of this thesis.

There have been numerous works of musical application of neural networks. Some examples are [19], [5], and [14]. In particular, two works that are closely related to the present study are presented in [10]. In [10], the authors present two neural networks, called HARMONET and MELONET. HARMONET produces and recognizes harmonization of chorales according to the styles of specific composers. Harmonization of a chorale has been dealt with in a previous section. HARMONET produces harmonization of

chorales on the level of an improvising organist [9]. Given that the style of the chorale harmonization is from one of three composers, HARMONET correctly identifies the composer of the chorale harmonization in 66 out of 68 cases. HARMONET takes input from a window; that is, for the output at sonority n, the input includes melodic information from sonorities $n-1$ to $n+1$ inclusive, as well as information on the harmonic function from sonorities $n-3$ to $n-1$ inclusive. This idea of a window is also used in this thesis. MELONET composes melodic variations to a voice of a chorale. Melodic variations is a technique musicians use, in which a simple melody is embellished and decorated with other notes, making the melody sound more interesting. The structure of the original melody is not changed. As deciding that how well a melodic variation sounds is subjective, an objective result is that the output of MELONET adheres well to the underlying harmonic context. One weakness of the network is that, due to its structure, it is unable to compose melodic variations where some of the notes are chromatic (i.e. not belonging to the current key).

A general key-finding algorithm is described by Krumhansl [13]. The algorithm works as follows. The input to the algorithm is a 12-dimensional vector; each of the 12 values represents the total durations (in number of beats) of a different pitch within an octave in a given musical passage. (Recall that an octave is divided into 12 equal intervals.) Enharmonically equivalent pitches are not distinguished. This input is correlated with 24 pre-defined 12-dimensional vectors, for each of the 12 major and 12 minor keys. Each of these pre-defined vectors describes the extent that each of the 12 pitches in an octave relate to the key this vector represents. Naturally, one of the

ways to use the results of the correlations is to pick the pre-defined vector that has the highest correlation with the input vector, and to state that the key that the pre-defined vector represents is the key of this musical passage. In terms of performance, in one test, the algorithm was given as input the first 4 tones of each of the 48 Preludes of J.S. Bach. Except for 4 cases, the algorithm was able to determine the correct overall key of the piece. In another test, one of the aforementioned Preludes was analyzed in its entirety. A key was assigned in a per-measure basis, for every measure in the piece. The input was taken from the current measure, the previous measure, and the subsequent measure. One extension to this algorithm, in the context of harmonic analysis, would be to increase the precision of analysis. Ideally, a key should be assigned in a per-sonority basis, so that if modulation occurs in the middle of a measure, or even in the middle of a beat, the algorithm can detect it. This thesis finds keys in a per-sonority basis.

A root-finding algorithm is described by Temperley [21]. In this algorithm, no information about key is found; instead, for every sonority, it outputs the letter name of the root of the chord that fits the most. No information about the quality of the chord is found, however. This is a simpler problem than the problem this thesis attempts to solve. For example, a sonority labeled "G" by this algorithm is equivalent to both V of C major and IV of D major. This algorithm uses five preference rules to determine the output. The author is "pleased with the results." However, there are some places where the algorithm performs poorly. For example, the algorithm has no notion of diminished chords. This illustrates the shortfall of a generic rule-based algorithm: there would most likely be corner cases that

defy the rules and cause the algorithm to fail. In contrast, a trained neural network has a better chance to cover more cases, given that it has had sufficient examples to train with.

An algorithm described by Taube [20] performs tonal analysis. Tonal analysis, as defined in [20], outputs the same type of information as the harmonic analysis performed in this thesis. However, the outputs represent different things. Tonal analysis does not imitate a human expert in trying to figure out what key every sonority belongs to. Instead, the algorithm searches throughout the piece for places of evidence of dominance of certain pitches. These places are called tonal centers. Then, between these tonal centers, the algorithm tries to analyze the piece according to the nearby tonal centers. The difference in the outputs between tonal and harmonic analysis is that in tonal analysis, the key appears to change more often than in harmonic analysis. Whereas a sonority may have a secondary function (e.g. V/V of C major) in harmonic analysis, the same sonority would have experienced a sudden key change (e.g. V of G major) in tonal analysis. Tonal analysis does not analyze as a human would; only harmonic analysis does. In a test suite of 10 J.S. Bach chorales that provide particular analytical challenges, this algorithm for tonal analysis produced essentially correct results.

Two works that precisely perform harmonic analysis are Maxwell [17] and Winograd [22]. In Maxwell [17], the algorithm consists of 55 rules. Of these, 36 are used to determine which sonorities are chords; the rest of the rules determine the key of every chord. Only chords are analyzed; the sonorities between chord A and chord B would receive the identical analysis as either chord A or B. The performance of this algorithm is generally good. One area

where the algorithm performs poorly is that it fails to take melodic patterns into account. Melodic patterns are helpful in harmonic analysis. Another problem area is that in the examples given, modulation seems to occur a little bit early. According to the rules of harmonic analysis, when there is a modulation, if there is a chord at the transition point that belongs to both the old and new key, then that chord is called a *pivot chord* [12]. However, the pivot chord at some places has turned into a pivot function lasting as long as more than two measures. In one instance, the transition from the old to the new key took more than two measures and involved four different chords. The algorithm in this thesis assigns only one pivot chord at the point of modulation.

The approach taken by Winograd [22] was to describe a grammar for tonal harmony. As can be imagined, the tonal harmony language is highly complex and ambiguous. For any given passage, many possible ways of parsing are possible. In short, the most meaningful way of parsing the passage is taken to be the correct way. To give a highly contrived example: the progression VI→V in C major is equivalent to II→I in G major, but the first progression (VI→V) is more meaningful, according to the rules of harmonic analysis. Since the number of ways of parsing grows exponentially with the length of the piece, the paper describes heuristics to keep the possibilities to a reasonable number. One main problem with this approach of performing harmonic analysis is that the melodic ideas are completely ignored, and sometimes these ideas are necessary in determining the correct analysis. The lack of melodic information sometimes result in analysis that is overly complicated in order to achieve a "meaningful" parsing. The approach in this

thesis uses melodic information since the actual notes are the inputs to the neural networks. In Winograd [22], the notes are present at first, but they are lost when they are parsed into chords, before the system decides which chord progressions are most meaningful.

2.4 Summary

This Chapter has discussed the background necessary to understand the rest of the thesis; namely, the concepts of harmonic analysis and neural networks. This Chapter also presented a literature review of topics related to either neural networks or harmonic analysis.

Chapter 3

Methods

This Chapter is broken down as follows. Sections 3.1 to 3.5 discuss the methods relevant in a global sense; namely, the type of music chosen, the types of networks chosen, the particular Bach chorales chosen, the simulation tool used, and an overview of network architecture. Sections 3.6 to 3.9 discuss the details of each of the four sub-networks, namely Key Network, Root Network A, Root Network B, and Quality Network. Section 3.10 gives an algorithm for determining pivot chords. Section 3.11 tells how the sub-networks are linked together.

3.1 Type of Music Chosen

The type of music chosen for harmonic analysis was J.S. Bach chorales. There are three reasons for this choice. First, they are relatively easy to analyze, and hence provide a good starting point. Music theory students often learn harmonic analysis through Bach chorales. Second, they are stylistically con-

sistent. This means that the neural networks only need to learn one style. Third, they are abundant. Bach harmonized almost 400 chorales. This provides ample training and testing data.

3.2 Types of Networks Chosen

As mentioned in the Background section, the feed-forward and Elman networks were chosen for use in this thesis. All networks have one layer of hidden units. For the feed-forward networks, the transfer function used in both the hidden units as well as the output units is the logistic sigmoid. For the Elman networks, the transfer function used in the hidden units is the hyperbolic tangent. The hyperbolic tangent was used instead of the logistic sigmoid because training with the former generalized much better than training with the latter. The transfer function for the output units is the logistic sigmoid.

3.3 Chorale Selection

Twenty J.S. Bach chorales were randomly selected as training data to the neural networks. They are numbers 5, 28, 33, 34, 44, 54, 64, 65, 75, 77, 79, 80, 111, 112, 126, 128, 159, 166, 192, and 378 from the Kalmus Edition [4]. There are a total of 1507 sonorities in these 20 chorales.

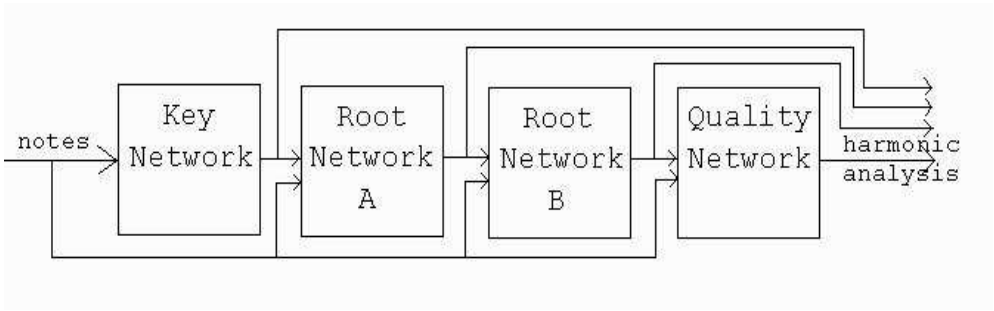

Figure 3.1: Overall network architecture block diagram. The entire network is divided into four smaller Networks: Key Network, Root Network A, Root Network B, and Quality Network. Analysis begins with the Key Network taking the notes information; it ends when the Quality Network gives its output.

3.4 Simulation Tool

MATLAB version 6.0 and the associated Neural Network Toolbox version 4.0 were used for training, testing, and simulating the neural networks.

3.5 Overview of Network Architecture

Figure 3.1 is a block diagram of the network architecture of this thesis.

The Key Network takes the notes of the current and surrounding sonorities and produces the key and mode of the current sonority. The Root Network A takes the notes of the current and surrounding sonorities and the output of the Key Network and produces the root that the current sonority tonicizes, if applicable. The Root Network B takes the notes of the current and surrounding sonorities and the outputs of Key Network and Root Network A and produces the root of the current sonority. The Quality Network takes the notes of the current and surrounding sonorities and the outputs of

Key Network, Root Network A, and Root Network B and produces the quality of the current sonority. Each of the four Networks is trained separately (i.e. each receives the correct input, not input from the output of a previous Network in the chain).

3.6 Key Network

3.6.1 Details of Training Inputs

3.6.1.1 Transposition of Chorales

The 20 chorales were written in different keys. Some keys occurred more often than others. If the chorales were fed as-is into the neural networks for training, the result would be that the keys occurring more would receive favourable treatment compared to the keys occurring less. To ensure that all keys are treated equally, each chorale is transposed 12 times, so that each chorale is found in the same 13 keys. Since there are twelve tones in an octave, the 13 keys cover all tones, with one tone covered by two enharmonically equivalent keys. Almost all, if not all, of the Bach chorales exist in one of these 13 keys. The number of elements in the training set of the Key Network is $1507 \times 13 = 19591$.

3.6.1.2 Parameters of Training Inputs

The Key Network has the task of determining the key of a particular sonority. It is given the same information as a human analyst would have: the notes of the sonority and those surrounding it; the metrical importance of the

sonority; and whether the sonority is at a cadence point. In Bach chorales, a cadence point indicates the end of a phrase.

3.6.1.3 Input Window Size and Position

It is clear that the key of a sonority cannot be determined from that sonority alone. Context is needed. However, how much context is necessary? Would too much context confuse the networks and degrade the performance? Context consists of sonorities from ahead and behind. Would having the input window centered on the current sonority be the best? Or would shifting the window front or back be better? Different input window sizes and positions were tried. For labeling purposes, the window size and position are referred to by the symbol $\alpha p \beta n$, standing for inclusion of α previous sonorities and β next, or subsequent, sonorities, in addition to the current sonority. Various window sizes from $7p7n$ to $11p11n$ were tried.

3.6.1.4 Inclusion of Metrical Information

Would including the metrical information of the sonorities inside the input window be beneficial in determining the key? It is initially thought that inclusion of metrical information would be beneficial because the metrically weak sonorities would be less important than the metrically strong sonorities in determining the key. It is highly likely that a metrically weaker sonority would belong to the same key as the preceding metrically stronger sonority (e.g. the sonorities are within the same beat). Both inclusion and exclusion of metrical information were tried. The symbol for inclusion of this scheme is DUR.

3.6.1.5 Inclusion of Cadential Information

Would including the cadential information of the sonorities inside the input window be beneficial in determining the key? It is initially thought that inclusion of cadential information would be beneficial because the key at the cadence point is very clearly heard and is always the same as the key at the preceding sonority. Both inclusion and exclusion of cadential information were tried. The symbol for inclusion of this scheme is CAD.

3.6.1.6 Coding

Each training input vector is split into n equal parts, n being the total number of sonorities in the input window. For each of the n parts, the coding scheme is as follows. The notes of the sonority in question are coded in a 4-of-m scheme, where $m = 27$ is the total possible number of notes. (Recall there are only 12 actual notes; the rest are enharmonic equivalents. In theory, for any given note, there is an infinite number of enharmonic equivalent notes. For the keys the chorales were transposed into, the 27 notes incorporate all the notes of the chorales that will be encountered.) This means that for each possible note, if it appears in the sonority, the corresponding position of the vector is assigned a value of 1 (except where it sometimes received a value of 2; this is described below in Section 3.6.1.7); otherwise, the corresponding position of the vector is assigned a value of 0. This means that if a note appears twice in the sonority, the corresponding position in the vector is still assigned 1. An alternative is to increment the value every time the note appears. The symbol for this alternative scheme is DT (standing for doubling-tripling). Both ways were tried. Another alternative (that was

not attempted) is to represent each voice in a 1-of-m scheme. This was not attempted because this would represent a fourfold increase in the length of each training input vector. This would take too long to train, given time constraints. If the input window stretches further back than the beginning of the piece or further ahead of the end of the piece, the respective "empty" sonorities are coded with all 0's. The metrical information of the sonority is coded as follows. All sonorities falling at the beginning of any beat receive a value of 3; those falling at the middle of any beat receive a value of 2; those falling at the quarter-point or third-quarters point receive a value of 1. For the cadential information, if the sonority is at a cadence point, it is assigned a value of "1,0"; otherwise, it is assigned a value of "0,1".

3.6.1.7 Different Weights in Different Voices

It is possible that some voices are more important in determining the key than other voices. Therefore, it was interesting to try coding the inputs in such a way that puts more weight on certain voices. As a starting point, it was decided to try two different weight values: 1 and 2. Since there are four voices, and having the four voices assigned all 1's or all 2's are equivalent, there are in total $2^4 - 1 = 15$ different combinations of weights. This scheme is labeled with the first letter of the voice(s) that is/are assigned a weight of 2. For example, if the *s*oprano and *a*lto voices are assigned a weight of 2, then the symbol is 'sa'. This scheme yielded some interesting results, which will be presented later in Chapter 4. Note that this scheme was not used in conjunction with the scheme where the value is incremented every time the note appears in the sonority (DT).

3.6.1.8 Other Alternate Coding Schemes

Two other coding schemes dealing with the notes of the sonorities were tried. One was to take only the sonorities with metrical level 3 and ignore all other sonorities with smaller metrical levels. In the Bach chorales, this translates to taking all the sonorities occurring at the beginning of the beat and ignoring all others that follow in the rest of the beat. It was assumed that the key is the same for the entire duration of a beat. This scheme is labeled with a symbol of QRT. The other coding scheme was similar, except that instead of ignoring the sonorities in the rest of the beat, they are superimposed on the sonority at the beginning of the beat. This scheme is labeled with a symbol of QRTMERGE.

3.6.2 Hidden Units

Values from 10 to 160 in increments of 10 were tried for the Key Network. Different values were tried in order to locate approximately the minimum number of hidden units required to learn the task at hand relatively well. It is expected that the accuracy will rise as the number of hidden units increases, up to a certain number of hidden units. After this, as the number of hidden units continues to increase, the accuracy will either remain the same or even decrease slightly.

3.6.3 Details of Training Output

The training output vector is coded in a 1-of-m scheme, where $m = 35$ is the total number of keys considered by the Key Network. The keys are

distinguished by modes. Thus, C major is distinguished from C minor, and each is represented at a separate position. Alternately, another attempt is to split the Key Network into two parts, A and B. Key Network A takes the notes of the sonorities as input and outputs the mode (in a 1-of-2 scheme) of the current sonority. Key Network B takes the notes of the sonorities and the output of Key Network A as input and outputs just the key of the current sonority.

3.6.4 Method of Training

3.6.4.1 Training Function

The primary training function used is gradient descent with adaptive learning rate and momentum. This is implemented by the function TRAINGDX inside MATLAB Neural Network Toolbox. Another training function that was tried was scaled conjugate gradient (TRAINSCG inside MATLAB Neural Network Toolbox).

3.6.4.2 Number of Epochs Trained

For TRAINGDX, training was stopped and restarted every 500 epochs. Initial results showed that performance almost always decreased after 1500 epochs. Therefore, all the networks were trained for up to and including 1500 epochs. It seemed that MATLAB was unable to remember the learning rate and momentum constant after training was stopped. When training restarts, the learning rate and momentum constant are reset to their default values. It was necessary to stop training in order to test the network. There

was a function in MATLAB which allowed the user to specify a validation set, and MATLAB would test the network after every epoch against this validation set, and if performance decreases for a user-defined number of times consecutively, training would stop. However, given the large amount of time that the network already takes to train, and that it did not seem that there was much gain in using this functionality, it was not used. For TRAINSCG, training was done for 250 and 500 epochs.

3.7 Root Network A

3.7.1 Details of Training Input

After the Key Network has figured out the key and mode of every sonority in the piece, the next step is to determine whether a sonority is tonicizing another chord. Compared to the Key Network, this network (and those following) is much smaller in size. This is because the key has been determined, and everything from now onwards is determined relative to the key, so therefore all the training input can be transposed into two common keys: one for major (C major), and one for minor (C minor). It is no coincidence that both keys are of the same letter name. It is done this way to reduce the dimensionality of the input space and to improve generalization. The dimensionality is reduced because these two keys share many notes in common; it improves generalization because many chords in the two scales share the same root. For example, the V chord of C major and V chord of C minor share the same root, G.

So the training input to Root Network A consists of the notes of the

current sonority and surrounding sonorities transposed to C major or C minor, depending on the mode. The window size for this network (and those following) is also much smaller in size. This is because the determination of tonicization of another chord is very much local in nature. The window sizes $0p0n$ (meaning just the current sonority is taken as input), $1p1n$, and $2p2n$ were tried. There are cases where the key changes somewhere inside the input window (this is referring to before the transposition to C). The key of the current sonority is taken to be the key of the entire input window, before the entire input window is transposed to C. The other alternative is to leave the key of each sonority as-is. While this approach would probably reduce the dimensionality of the input space, the input window is transposed from two different keys and so is not a true transposition from the original version. As a result, some important information might be lost.

3.7.2 Hidden Units

Values from 10 to 60 in increments of 10 were tried for Root Network A.

3.7.3 Details of Training Output

The training output is coded in a 1-of-7 scheme. The 7 possible outputs are I, II, III, IV, V, VI, and ♭VII (one semitone below VII). The output I is for the sonority that is not tonicizing any chord (this actually represents over 90% of all outputs). The other outputs represent the chord of the scale the sonority is tonicizing. For example, the output V means that the sonority is tonicizing the V chord of the given key.

3.7.4 Method of Training

The training function TRAINGDX was used. The numbers of epochs attempted were 500, 1000 and 1500. The network performing the best will be shown.

3.8 Root Network B

3.8.1 Detail of Training Input

The training input is identical to that for Root Network A, with one exception. If the output of Root Network A for the sonority in question is not I, then the key of this sonority is not solely determined by the Key Network; it also needs to take the output of Root Network A into account. For example, if the output of the Key Network for the sonority in question is A major, and the Root Network A's output for this sonority is V, then the key of this sonority is V of A major, which is E major. The window sizes/positions attempted are $1p1n$, $1p2n$, $2p1n$, and $2p2n$.

3.8.2 Hidden Units

Values from 20 to 60 in increments of 10 were tried for Root Network B.

3.8.3 Detail of Training Output

The training output is coded in a 1-of-9 scheme. The 9 possible outputs are I, II, III, IV, V, VI, VII, ♭VI (one semitone below VI), and ♭VII. This represents the sonority's harmonic function within the given key. For example, if the

key of a sonority is D major as determined by the Key Network, and the output of Root Network A is V, and the output of Root Network B is V, then it is said that this sonority's harmonic function is V/V in D major. As another example, if everything is the same as the previous example, except that the output of Root Network A is I, then it is said that this sonority's harmonic function is V in D major. It is possible to say V/I in D major, but this is redundant (since I in D major is D major) and hence omitted.

3.8.4 Method of Training

The training function TRAINGDX was used. The numbers of epochs attempted were 500, 1000, and 1500. The network performing the best will be shown.

3.9 Quality Network

3.9.1 Detail of Training Input

The training input is identical to that for Root Network B, except for two things. First, the key of the current sonority is not taken as the key of the entire input window. Each sonority is transposed according to its own key. Second, the output of Root Network B is included in the input. Now, it is actually necessary to transpose each sonority according to its own key because the output of Root Network B, the harmonic function, is relative to the key of the sonority. Leaving out the output of Root Network B is not a good idea because it helps in the determination of the quality of the sonority.

Window sizes/positions of $1p1n$, $1p2n$, and $2p1n$ were attempted.

3.9.2 Hidden Units

The values from 20 to 60 in increments of 10 were tried for Quality Network.

3.9.3 Detail of Training Output

The training output is coded in a 1-of-9 scheme. The 9 possible qualities of a sonority are major, minor, augmented, diminished, dominant 7th, diminished 7th, major 7th, minor 7th, and half-diminished 7th. These are the same qualities listed in Table 2.2 on page 13.

3.9.4 Method of Training

The training function TRAINGDX was used. The numbers of epochs attempted were 500, 1000, and 1500. The network performing the best will be shown.

3.10 Determination of Pivot Chord

The algorithm for determining pivot chords is fairly straightforward. Assume that sonority n is in key X and sonority $n+1$ is in key Y. Then, sonority $n-k$ to sonority n inclusive are considered pivot chord if all of these hold: 1) these sonorities are of the same harmonic function in key X; 2) these sonorities, if analyzed in key Y, have a harmonic function in key Y. The pivot chord receives functional analysis in the context of both key X and Y. In Bach

chorales, the pivot chords are very short: $k = 0$ or 1 are common values. Sometimes, if the key changes abruptly, there may not be a pivot chord at all. As this is a simple algorithm and has no bearing with the testing results from the neural networks, it will not be further discussed.

3.11 Putting Them Together

The input to the first Network is the set of ordered notes of the piece. At the end, the harmonic analysis is complete, including key, harmonic function, and quality for every sonority. During testing, the output of each Network is manipulated as follows, before being passed on to the next Network: the largest value of the output vector is set to 1, and the rest of the values are set to 0.

3.12 Summary

This Chapter has presented the methods used to perform harmonic analysis. It has been seen that the entire network was divided into Key Network, Root Network A, Root Network B, and Quality Network, each trained to perform a specific part of harmonic analysis on Bach chorales. Each Network (except Key Network) requires input from other Network(s). Different coding schemes are tried for the Key Network. The two types of network architecture used are feed-forward and Elman.

Chapter 4

Results

This Chapter is divided as follows. Section 4.1 gives information on what music the test suite consists of. Section 4.2 presents results for Key Network. Section 4.3 presents results for Root Network A. Section 4.4 presents results for Root Network B. Section 4.5 presents results for Quality Network.

4.1 Test Suite

The networks were tested with a test suite of 18 Bach chorales. They are numbers 7, 9, 26, 30, 32, 50, 57, 91, 93, 138, 154, 175, 189, 201, 206, 211, 224, and 284 from the Kalmus Edition. Of these, 8 were also chorales chosen to be tested in [20]. These chorales were chosen because they "provide particular analytical challenges." [20, page 29] The other 10 chorales were randomly selected. These 18 Bach chorales represent 1362 sonorities in total.

4.2 Key Network

Figure 4.1 shows the results obtained by putting the test suite through networks that include/exclude cadential and metrical information. The window size/position for all was fixed at $8p8n$. The type of network was feed-forward (FF). The number of hidden units for all networks in this figure is 90. The voice weight scheme is none, meaning all four voices received equal weight. From this graph, concerning the point of including or excluding cadential and metrical information, it can be seen that the network that neither includes cadential nor metrical information performed the best. For the other networks tested in this section, they all include both cadential and metrical information because many networks were already trained with the inclusion of both cadential and metrical information before the other schemes were thoroughly tested. Figure 4.1 also shows the results of splitting the Key network into two, one to handle the key and the other to handle the mode. As can be seen, it is clearly not a good idea. The accuracy rate is down sharply. This suggests that the learning of key and mode is interdependent. As well, the scheme of considering just the sonorities at the beginning of beats (QRT) or the scheme of merging sonorities from the same beat into one sonority (QRTMERGE) produces sub-par results. It is reasoned that QRT does not work well because a sizeable fraction of sonorities is gone, thus hurting the generalizing ability of the networks; it is thought that QRTMERGE does not work well because too much information is packed into one sonority, thus increasing the input space. The scheme of showing the possible doubling or tripling of the same note in the same sonority (DT) appears here to be slightly better than representing repeated notes as one note. The

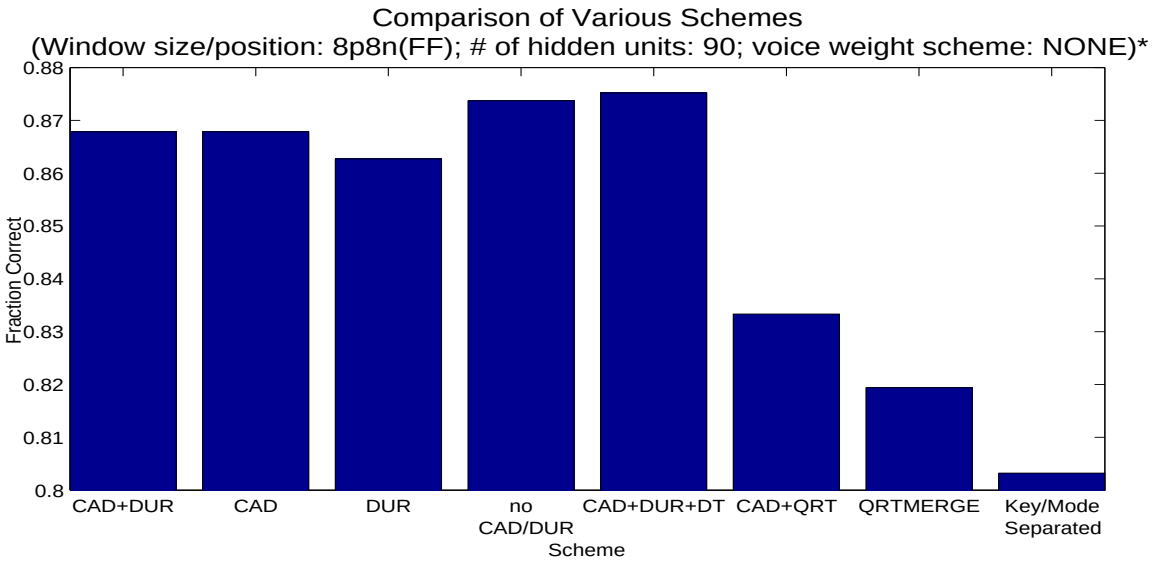

Figure 4.1: Comparison of Performance of Various Schemes of Key Network

networks that follow do not use the scheme DT because most of the networks were already trained with DT before the discovery that DT works slightly better.

Training using TRAINSCG was done on a $9p7n$(FF) network with 120 hidden units and schemes CAD and DUR included. After 250 epochs, the network performed at an accuracy of 86.6%. This, compared with TRAINGDX, which takes up to 1500 epochs to achieve the same level of accuracy, indicates that TRAINSCG converges faster than TRAINGDX. However, all other networks in this Chapter were trained with TRAINGDX because TRAINSCG was used at a late stage of development when the bulk of the training was already completed using TRAINGDX.

Figure 4.2 shows the results obtained by putting the test suite through networks that have the same window size/position ($9p7n$), are of the same

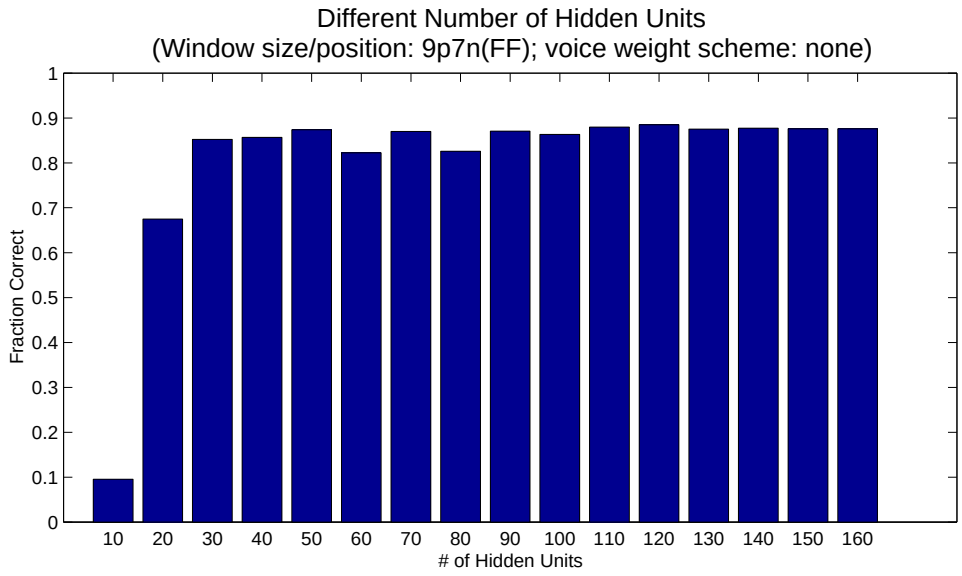

Figure 4.2: Performance of Different Number of Hidden Units of Key Network

type (FF) and voice weight scheme (none), but have different number of hidden units. From this figure, it can be seen that at 50 hidden units, the network's performance is almost as good or greater than the performance of the network at higher numbers of hidden units. Theoretically, a network with a larger number of hidden units can do no worse than one with a smaller number of hidden units when only training data is considered; however, it is easier for the larger network to overfit the training data and hence do worse than the smaller network on new testing data [6]. From this test, it seems that 50 hidden units are sufficient. The highest accuracy obtained was with 120 hidden units; in the networks that follow, they were all trained with 120 hidden units.

Figure 4.3 shows the results obtained by averaging the results of all 15 voice weight schemes for each window size/position. (The results for each

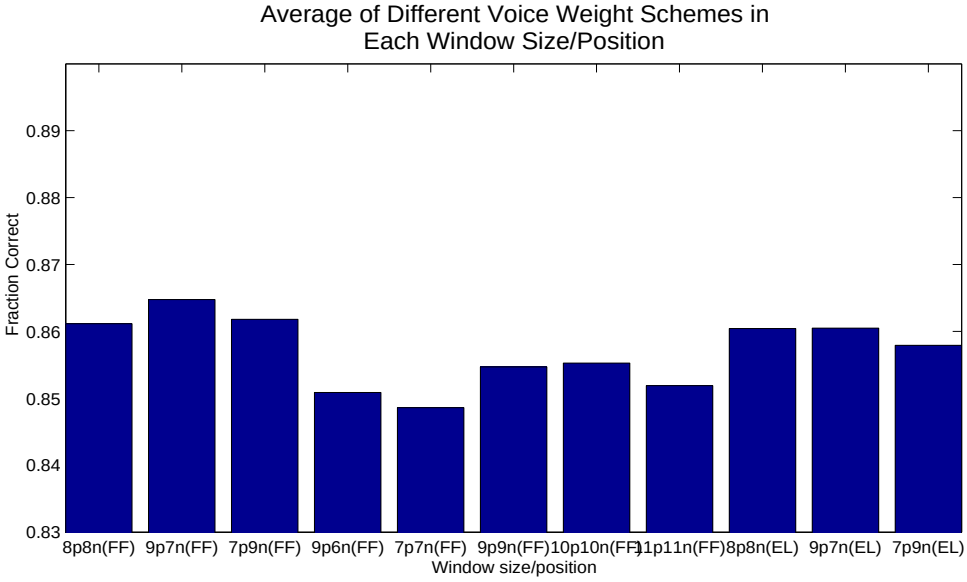

Figure 4.3: Average Performance of Each Window Size/Position

window size/position can be found in Appendix A.) The symbol (EL) means that the network is an Elman network. From the graph, the FF networks that had its window position centered on the current sonority (i.e. window size/position = $\alpha p \beta n$ where $\alpha = \beta$), $8p8n$ performed the best. Therefore, it seemed reasonable to attempt shifting the position around. From the graph, it can be seen that both $9p7n(FF)$ and $7p9n(FF)$ did better than $8p8n(FF)$, with $9p7n(FF)$ coming out on top. The reason that $9p7n$(FF) does slightly better than $7p9n$(FF) could be that past information is a slightly greater key-determinant than future information. Performance dropped when the window position was shifted to $9p6n$ (but, as shall be seen, these networks are valuable under a different context). The Elman networks gave comparable level of performance as their feed-forward counterparts of equal window size/position. It was initially thought that the Elman networks would per-

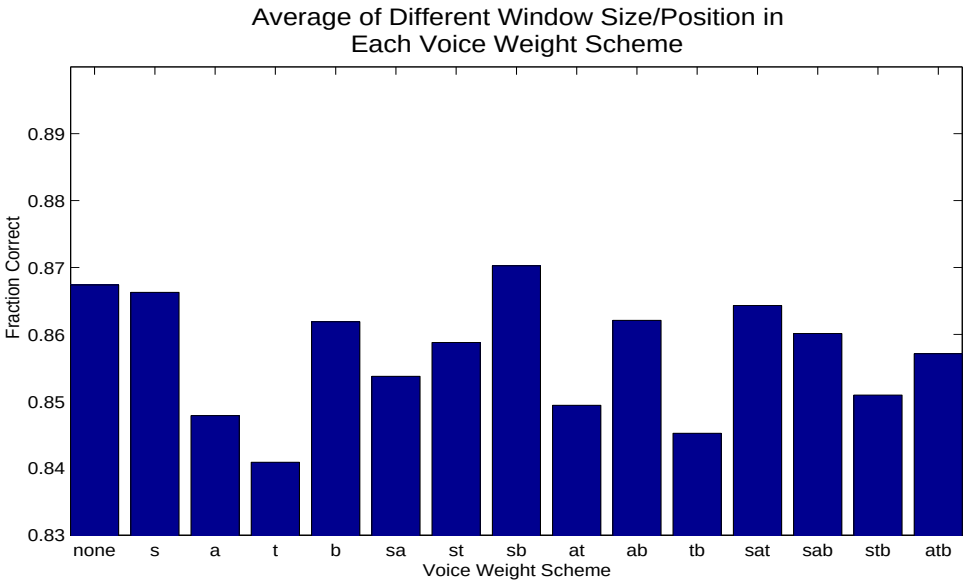

Figure 4.4: Average Performance of Each Voice Weight Scheme

form better than the feed-forward networks because of their ability to learn temporal patterns. The results do not indicate this. The reason could be that the temporal patterns are already learned by having such a large window size. It is speculated that a slightly smaller window size for the Elman networks would give slightly better results; this can be subject to further investigation.

Figure 4.4 shows the results obtained by averaging the results of all 11 window sizes/positions for each voice weight scheme. The fact that there are significant differences among the voice weight schemes strongly suggests that certain combinations of voices are better key-determinants than others. From the graph, it can be seen that the voice weight scheme 'sb' performs the best. From a musical point of view, this makes sense. The soprano line and the bass line, being the topmost and bottommost voice, respectively, are

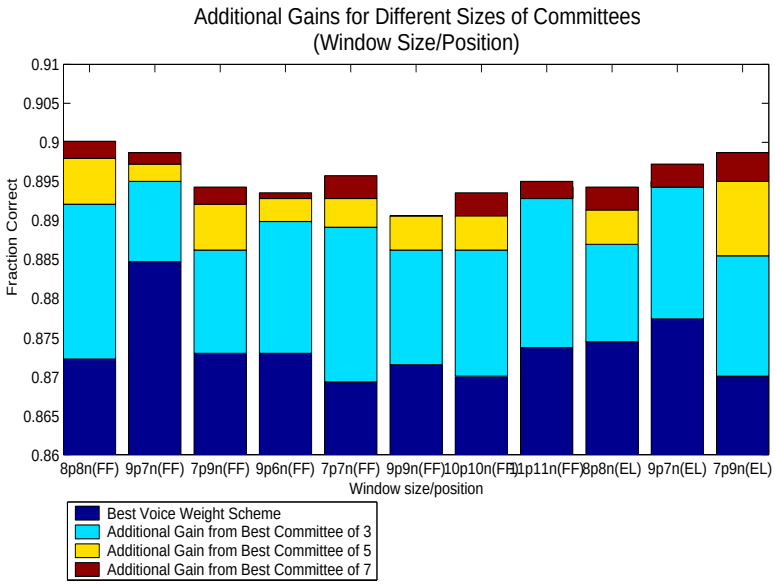

Figure 4.5: Performance Gain from Committees of Different Voice Weight Schemes of Same Window Size/Position

the two voices that are most easily heard. Hence, it would make sense for the composer (in this case, Bach) to place notes that emphasize the current key in these 'outer' voices rather than the 'inner' voices (i.e. alto and tenor), unless, of course, the composer writes in a style that gives the impression of unstable keys. But this is not the style of the Baroque-era Bach [16]. It would be interesting in the future to train the networks on the chorales of another composer, and see whether emphasizing the soprano and bass line also turns out to be the best.

Figure 4.5 shows the results obtained by placing networks with the same window size/position but different voice weight schemes into committees. Committees of 3, 5, and 7 were formed. Since there are a total of 15 networks per window size/position, the total number of possible committees of

n is $\sum_{k=1}^{n} \binom{15}{k}$. Note that this sum includes the combinations where more than one member of the committee comes from the same network (otherwise it would just be $\binom{15}{n}$). For $n = 7$, the sum amounts to 16383. For each n, the committee that performed the best is compared with the best performing single network for that window size/position. As can be seen, the best committee of 3 has a significant improvement over the best performing single network. For the committee of 5 and the committee of 7, their further improvement is marginal. For two window sizes/positions ($11p11n(FF)$ and $9p7n(EL)$), their best committee of 5 did not do better than their best committee of 3. For $9p9n(FF)$, its best committee of 7 did not do better than its best committee of 5. The best committee of 7 of $8p8n(FF)$ obtained an accuracy of 90.0%, the best overall in this figure.

A compilation of which voice weight schemes appeared in the best committees of 3, 5, and 7 is given in Figure 4.6, 4.7, and 4.8 respectively. A voice weight scheme receives $1/n$ point for each time that it appears in the best committee for a window size/position, if n committees achieved the highest accuracy. Clearly, the scheme 'sb' is involved in the best committees more than any other scheme. This indicates that the networks trained in this scheme not only achieve a higher accuracy than networks trained in other schemes; it also indicates that a lot of breadth (i.e. predicts those sonorities correctly where other networks do not) is covered with this scheme. A counterexample is the scheme 'none', which after 'sb' is the highest performing scheme on average (see Figure 4.4). However, as seen in Figures 4.6 to 4.8, few of these networks make it on the best committees. This means that these networks do not cover a lot of breadth.

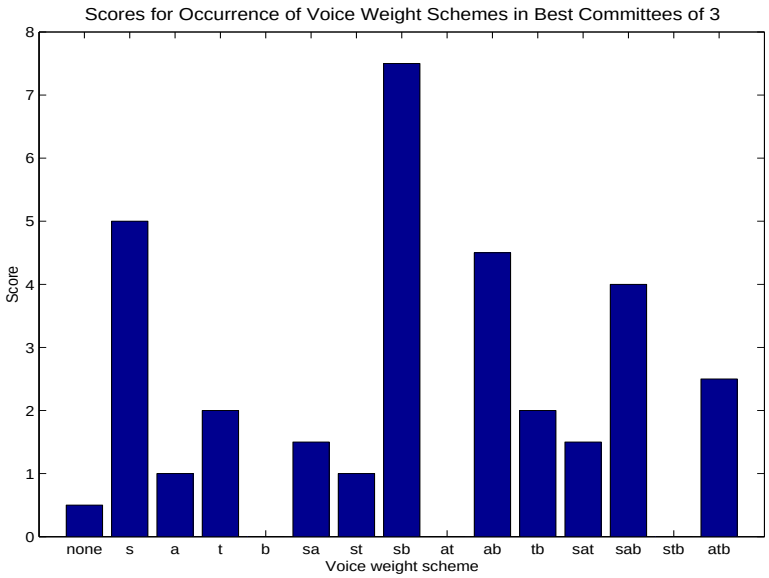

Figure 4.6: Scores for occurrence of voice weight schemes in the best committees of 3. The score is calculated as follows: given that n committees achieved the same highest accuracy, a voice weight scheme receives $1/n$ point for each time that it appears in the best committee for a window size/position.

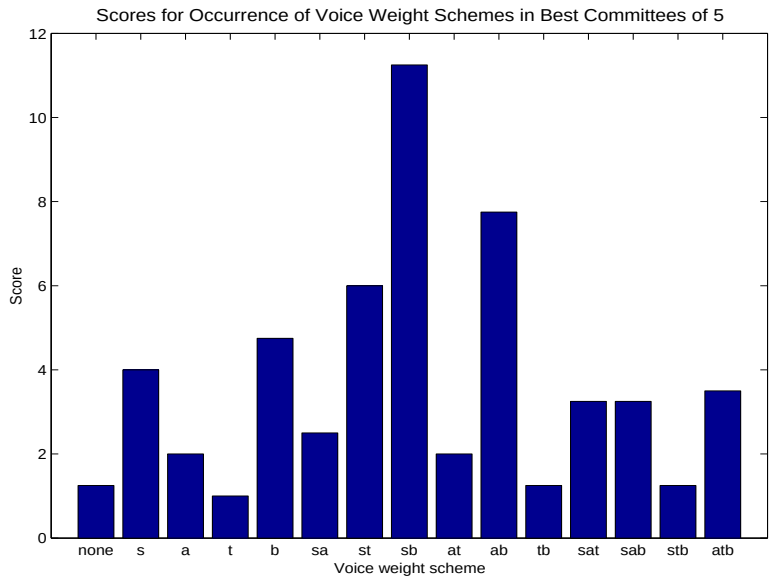

Figure 4.7: Scores for Occurrence of Voice Weight Schemes in the Best Committees of 5

It was mentioned in Chapter 2 that the output of the committee is obtained by taking the most frequently appearing largest output from the member networks. Early results had indicated that this form of the committee is better than that described in [6], which is taking the largest output of the average of the outputs of the member networks. However, late in the development process, it was found that the form of the committee described in [6] is generally superior. In the case of committee of 3 $7p9n$ networks, the best combination using the scheme from [6] is 0.5% better than the one currently used. A possible reason why early results had contradicted later results is that the early results were from forming a committee whose member networks had the same scheme of inputs but different initial weight values, while the later results were from forming a committee whose member networks had

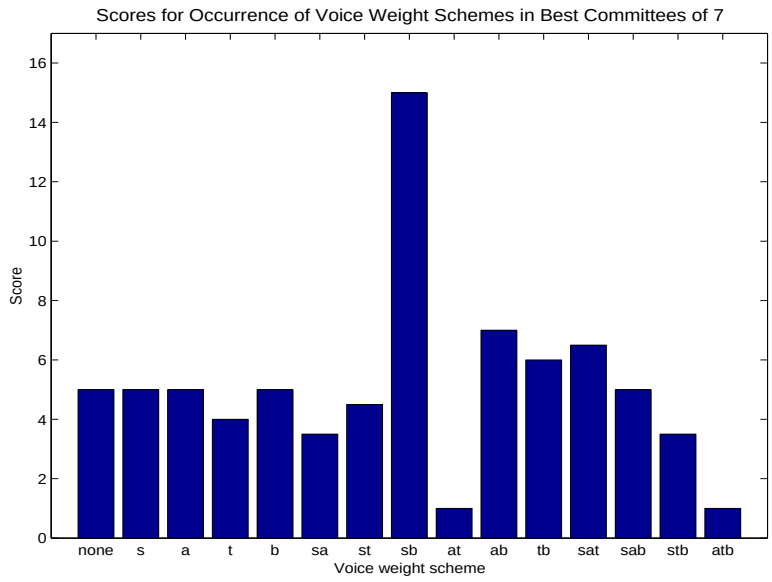

Figure 4.8: Scores for Occurrence of Voice Weight Schemes in the Best Committees of 7

different input schemes.

Figure 4.9 shows the results obtained by placing networks with the same voice weight scheme but different window sizes/positions into committees. As can be seen, the scheme 'sb' once again comes out on top, with an accuracy of 89.7% from its best committee of 7. The improvement of the best committees are measured with respect to the best performing window size/position for that voice weight scheme. The trend of significant improvement from best committee of 3 and marginal improvement from best committee of 5 and best committee of 7 remains. For the scheme 'sat', there was no improvement in its best committee of 5 over its best committee of 3.

A compilation of which window sizes/positions appeared in the best committees of 3, 5, and 7 is given in Figure 4.10, 4.11, and 4.12 respectively.

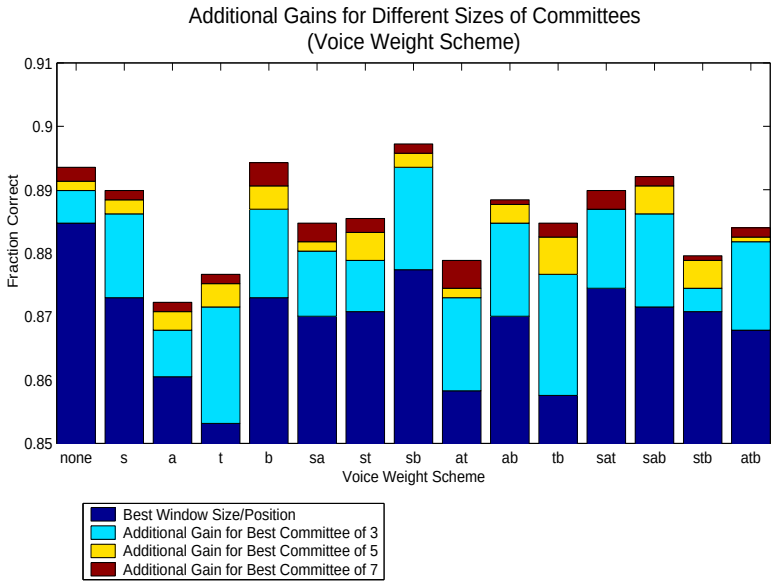

Figure 4.9: Performance Gain from Committees of Different Window Sizes/Positions of Same Voice Weight Scheme

The scoring is similar to that in Figures 4.6 to 4.8. A surprising result is that the window size/position $9p6n$(FF) did the best in all three sizes of committees. It is surprising because the average performance of $9p6n$(FF) networks is the second lowest amongst all window sizes/positions (see Figure 4.3). The reason seems to be that these networks cover cases other window sizes/positions do not. The $9p6n$ window position is the most unbalanced in terms of the number of previous sonorities and the number of subsequent sonorities. In the future, it may be fruitful to try other such unbalanced window sizes/positions, like $6p9n$, $10p7n$, or $7p10n$. The next highest scoring window size/position is $9p7n$(FF). This is expected because it is the highest scoring window size/position in Figure 4.3.

Additional gains to the accuracy were made when the best committees of

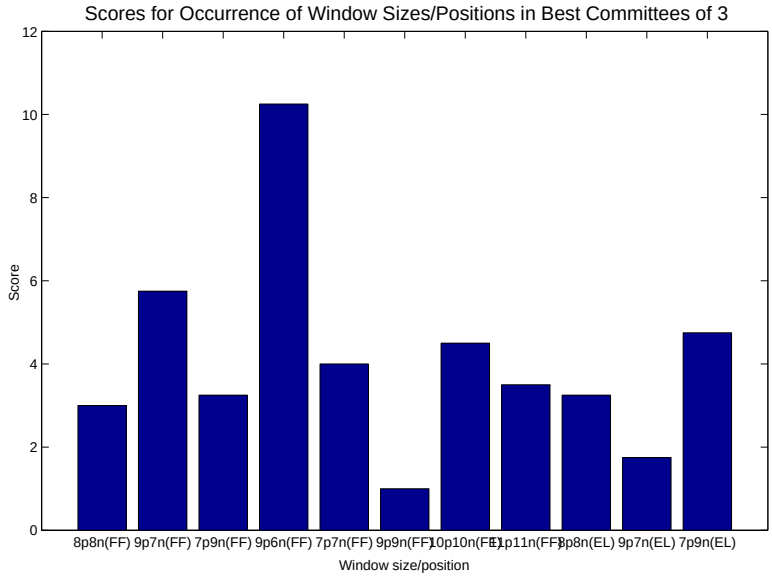

Figure 4.10: Scores for Occurrence of Window Sizes/Positions Schemes in the Best Committees of 3

7 in Figure 4.5 were further put into committees, thus forming committees of committees. The best committee of committees (henceforth referred to as *bestcom_key*) achieved an accuracy of 91.7%, thus improving upon the previous record of 90.0% by 1.7%. The best committee of committees formed from those in Figure 4.9 had a slightly lower accuracy of 91.4%. The best performing single network was the one with window size/position of $9p7n(FF)$ and voice weight scheme of none; it produced an accuracy of 88.5%. Thus, just by combining networks that were each trained a little differently (albeit by exhaustively trying every combination), the accuracy had gone up by more than 3%.

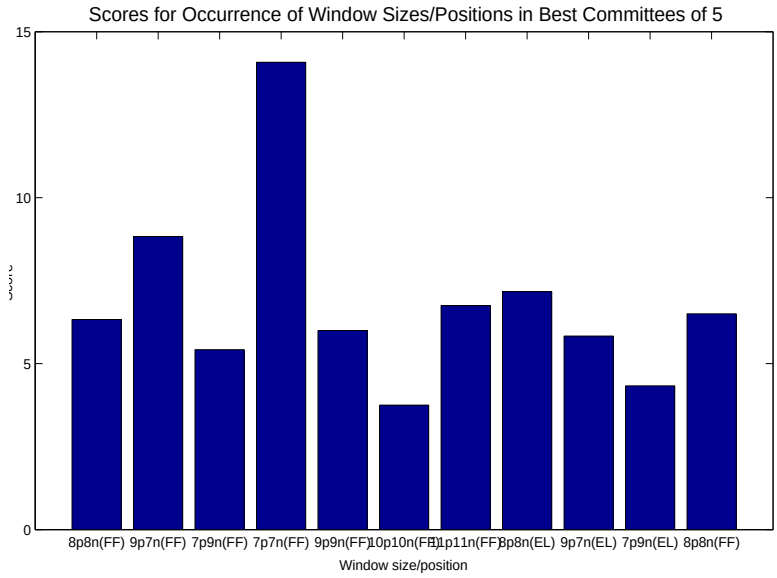

Figure 4.11: Scores for Occurrence of Window Sizes/Positions Schemes in the Best Committees of 5

4.3 Root Network A

The results in this section are obtained assuming the accuracy of Key Network is 100%. Figure 4.13 shows the performance of Root Network A when it was trained in 4 different window sizes/positions and with 6 different numbers of hidden units. For each combination of window size/position and number of hidden units, three networks were trained, each with different initial weights (this also applies to Root Network B and Quality Network, below). Figure 4.13 shows the best performing network. As can be seen, the networks trained with no context (i.e. window size/position of $0p0n$) perform the best. This is in agreement with the fact that tonicization is a local effect, so local that it involves only one sonority most of the time. As

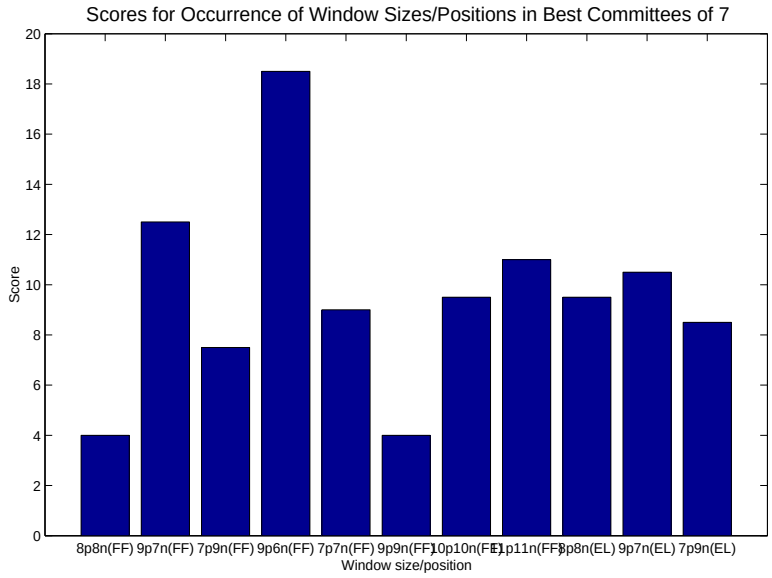

Figure 4.12: Scores for Occurrence of Window Sizes/Positions in the Best Committees of 7

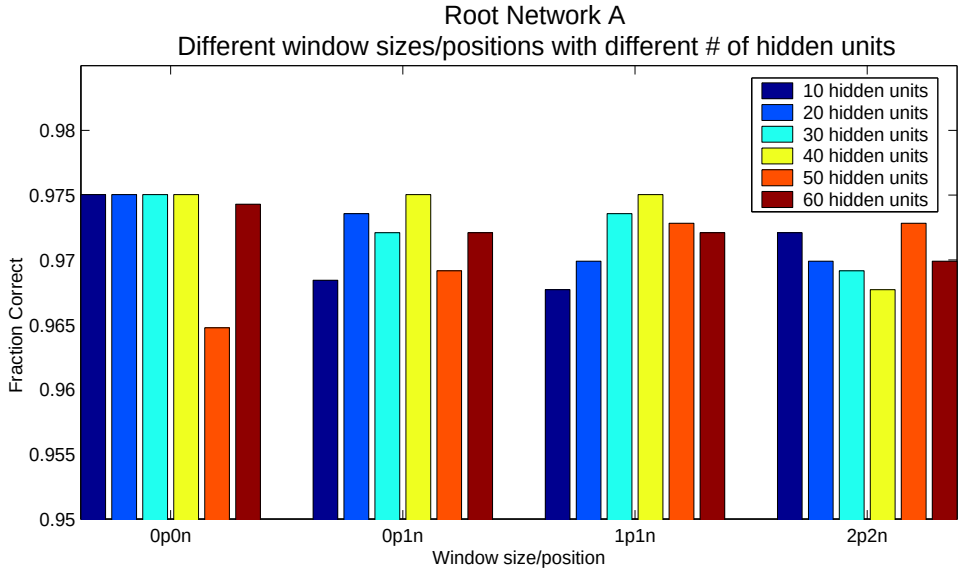

Figure 4.13: Performance of Root Network A

for the number of hidden units, as little as 10 hidden units is sufficient. The best result of 97.5% comes from six different combinations of hidden unit and window size/position. This might seem to be a great number; however, recall that over 90% (93.8%, to be exact) of the sonorities in the test suite are not involved in tonicization. Hence, it is just obtaining an increased accuracy of 3.7% over a naive algorithm which states every sonority is not involved in tonicization. A committee of committees, *bestcom_rootA*, was made in the following manner. First, a set of committees is made from all possible combinations of three of all the networks trained with the same window size/position and number of hidden units, for every possible combination of window size/position and number of hidden units. The best out of each set is further put into the next level of committees. Again, all combinations of three committees are tried. The best combination becomes *bestcom_rootA*. It yielded a 0.6% increase in overall accuracy. Further analysis of the performance of this network appears in Chapter 5.

4.4 Root Network B

Similar to the previous section, the results in this section are obtained assuming perfect accuracy of the Key Network and Root Network A. Figure 4.14 shows the performance of Root Network B when it was trained in 4 different window sizes/positions and with 5 different numbers of hidden units. The window size/position $1p1n$ seemed to be the best for this network. Also, at least 30 hidden units seemed to be needed, as three of the four cases with 20 hidden units had the lowest accuracy for their own window

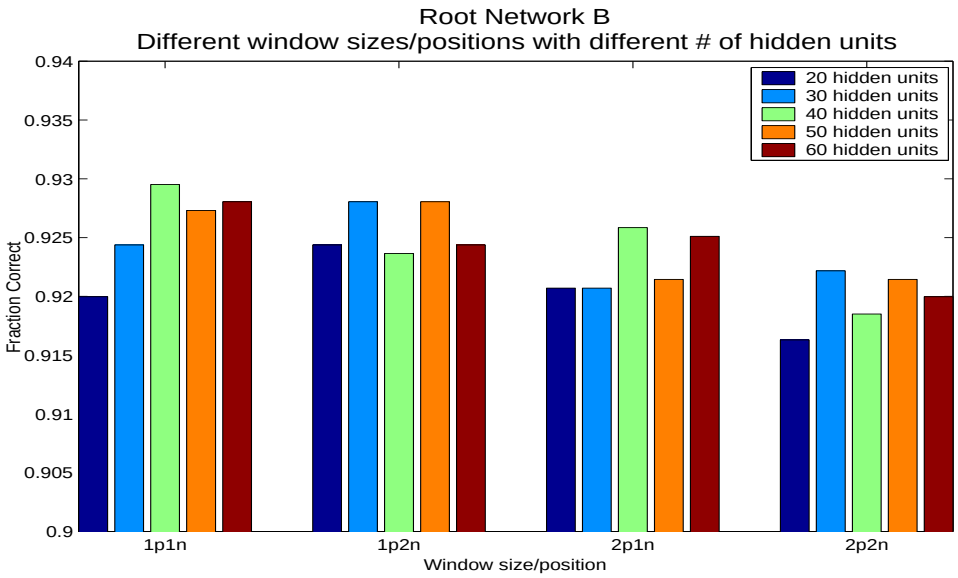

Figure 4.14: Performance of Root Network B

size/position. The best performing network had a window size/position of $1p1n$ and 40 hidden units; it obtained an accuracy of 93.0%. A committee of committees, *bestcom_rootB*, was made in the manner similar to that which made *bestcom_rootA*. It obtained a 1.2% increase in overall accuracy over the best performing single network. Further analysis of the performance of *bestcom_rootB* appears in Chapter 5.

4.5 Quality Network

The results in this section assume perfect outputs from the previous Networks. Figure 4.15 shows the performance of Quality Network when it was trained in 3 different window sizes/positions and with 5 different numbers of hidden units. There was a network trained in each of the 3 window

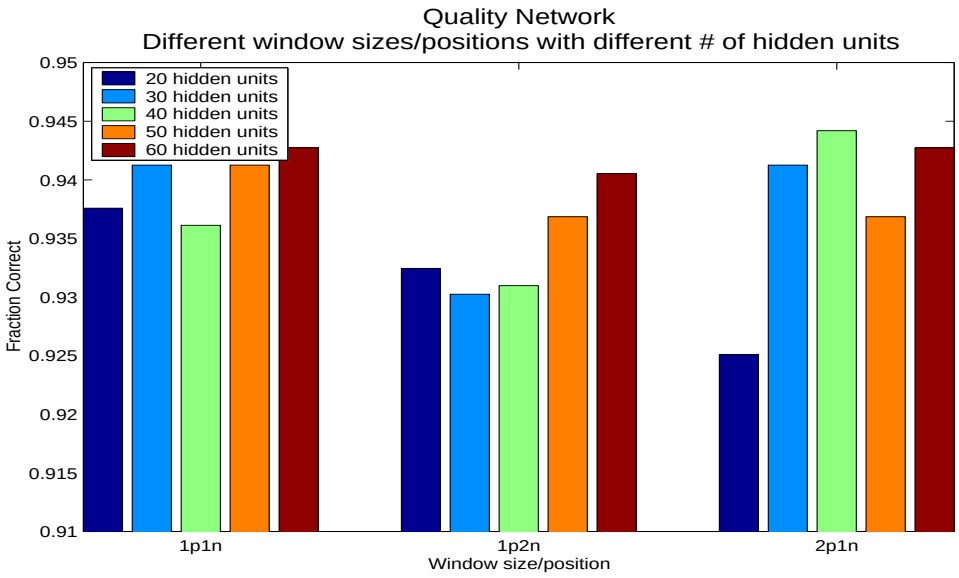

Figure 4.15: Performance of Quality Network

sizes/positions that did very well. From the graph, it is difficult to say which window size/position is the best. The best performing single network was the one with window size/position of $2p1n$ and 40 hidden units; it obtained an accuracy of 94.4%. The committee *bestcom_quality* was obtained in a similar manner to that which made *bestcom_rootB*. This committee performed a further 1.7% better in overall accuracy than the best performing single network. Further analysis of the performance of *bestcom_quality* appears in Chapter 5.

4.6 Summary

This Chapter has presented the results of testing each of the four Networks on unseen chorales. For Key Network, results for different coding schemes

were presented. In particular, it has been seen that the window size/position $9p7n$(FF) and the voice weight scheme 'sb' perform the best when compared with other window sizes/positions and voice weight schemes, respectively. Committees of networks were put together to achieve significant gains in some cases.

Chapter 5

Discussion

This Chapter is broken down as follows. Section 5.1 is an analysis of the incorrect predictions made by each of the four Networks. Section 5.2 gives the overall accuracy of the entire network. Section 5.3 is an analysis of a musical passage by another composer using the Key Network. Section 5.4 is a comparison of the performance of the network of this thesis with other published algorithms.

5.1 Analysis of Incorrect Predictions

5.1.1 Key Network

The best committee of committees (*bestcom_key*) achieved an accuracy of 91.7% in the test suite of 18 Bach chorales. For the 8.3% that were analyzed incorrectly, 3.7% involved modulation at a different place than where the author deemed it should, and a further 3.2% involved sonorities that were analyzed as being in closely related keys. Generally speaking, the exact place

of modulation is sometimes open to different interpretation. One person might view it as occurring at a certain place; another might think that it occurs at an earlier or later place. For the purposes of this thesis, every time modulation occurs in a piece, there is only one place for modulation where it would be deemed "correct." Nonetheless, analysis of the errors indicates that nearly half of the errors are due to modulation occurring at a different place than that allowed. This type of error should not be considered a severe one. The keys that are closely related to key X are 1) its relative major/minor; 2) the keys that differ from key X by only one in their key signatures and which are of the same mode as key X. If key X is major (minor), then its relative minor (major) is the minor (major) key that has the same key signature as key X. For example, the closely related keys of C major are A minor, G major, and F major. This type of error is genuine; however, it can be argued that, in these cases, the network's prediction is not far off from the correct answer. During training, the network learns to associate certain patterns of notes to certain keys. It is likely that the network learns to distinguish between different patterns of notes by the occurrence/absence and frequency of different notes. This is likely because keys are different by virtue of their usage of different subsets of the 12 notes in an octave, and also by how often certain notes in the subset are used (for example, the tonic note is most frequently used). It is reasonable to assume that the network learns this to a certain extent. The closely related keys of key X differ from key X by at most two notes. Let key Y be a closely related key of key X. Assume that the network has learned to associate a collection of patterns $P_x = P_{x1}, ..., P_{xn}$ with key X, and respectively $P_y = P_{y1}, ..., P_{yn}$ with key Y. During testing,

the inputs that resemble any pattern P_{xi} will be assigned the output key X. However, it is not unreasonable to assume that some of these inputs have an intended output of key Y, even though they contain P_{xi}. Essentially, this means that for some i and j, $P_{xi} = P_{yj}$. The network has not learned to distinguish between the two. However, since key X and Y are closely related and differ by at most two notes, arguably some of their patterns are quite similar also and difficult to distinguish.

Therefore, only 8.3% - 3.7% - 3.2% = 1.4% of testing inputs were assigned outputs that are seriously wrong. Another measure of merit for the Key Network is that for the members of *bestcom_key*, at least one member predicted the correct answer 95.6% of the time. This demonstrates the diversity of the members of *bestcom_key*.

5.1.2 Root Network A

The committee *bestcom_rootA* achieved an accuracy rate of 98.1%. Its performance breakdown is shown in Table 5.1. The roots are shown in decreasing order of frequency as they appear in the training set. As mentioned before, the vast majority of sonorities are not involved in tonicization (indicated by the root I). This makes it difficult for the network to learn the other roots. One way to increase further the accuracy of Root Network A is to make all roots equally frequent in the training set. This would mean picking some parts of a chorale for training and discarding the rest.

RootA	Total Training Sonorities	Total Testing Sonorities	# Correct	Accuracy (%)
I	1394	1277	1271	99.5
IV	30	31	22	71.0
V	27	26	24	92.3
III	19	5	3	60.0
VI	19	14	13	92.9
bVII	12	8	2	25.0
II	6	1	1	100.0
Total	1507	1362	1336	98.1

Table 5.1: Breakdown of performance of *bestcom_rootA*. A simple calculation shows that the prediction accuracy for non-tonicizing sonorities is 65/85 = 76%.

5.1.3 Root Network B

The committee *bestcom_rootB* achieved an accuracy rate of 94.2%. Further analysis revealed that an additional 2.0% of sonorities should also be considered correct. As this is analysis of music, there is a subjective part to it. Hence, although one answer is correct, another might also be. In this particular case, roots of sonorities can be different depending on the level of analysis. One may choose to analyze at a very high level to obtain a grand picture and flow of the entire piece of music. Alternatively, one may choose to analyze at a granular level to obtain root information of every single sonority. In this thesis, the chorales were trained with root information at a fairly granular level; during testing, a very slight variation on the level were allowed, hence the additional 2.0%.

Table 5.2 is an analysis breakdown of *bestcom_rootB*. The roots are ranked in decreasing order of frequency in the training set. As can be seen, the most frequently occurring roots in the training set achieve the highest accuracies in the testing set. The roots in the last four rows of Table 5.2 did not occur in the training set; so, it is natural that the network cannot recognize them.

5.1.4 Quality Network

The committee *bestcom_quality* achieved an accuracy rate of 96.1%. Further analysis (along the same line of reasoning as that for Root Network B) increased the rate to 96.4%. As an example, there is a sonority that can be labeled either as minor or as minor 7th. Table 5.3 is an analysis breakdown

RootB	Total Training Sonorities	Total Testing Sonorities	# Correct	Accuracy (%)
I	498	480	472	98.3
V	492	419	416	99.3
IV	175	162	155	95.7
II	106	92	88	95.7
VII	94	98	86	87.8
VI	56	49	44	89.8
III	45	29	23	79.3
bVI	30	20	18	90.0
bVII	11	9	8	88.8
bII	0	1	0	0.0
#VI	0	1	0	0.0
#II	0	1	0	0.0
Gr6	0	1	0	0.0
Total	1507	1362	1310	96.2

Table 5.2: Breakdown of Performance of *bestcom_rootB*

Quality	Total Training Sonorities	Total Testing Sonorities	# Correct	Accuracy (%)
Major	731	656	653	99.5
Minor	402	354	351	99.2
Dominant 7th	194	161	155	96.3
Diminished	74	65	61	93.8
Minor 7th	39	45	39	86.7
Diminished 7th	28	27	23	85.2
Half-diminished 7th	20	42	29	69.0
Major 7th	18	8	2	25.0
Augmented	1	4	0	0.0
Total	1507	1362	1313	96.4

Table 5.3: Breakdown of Performance of *bestcom_quality*

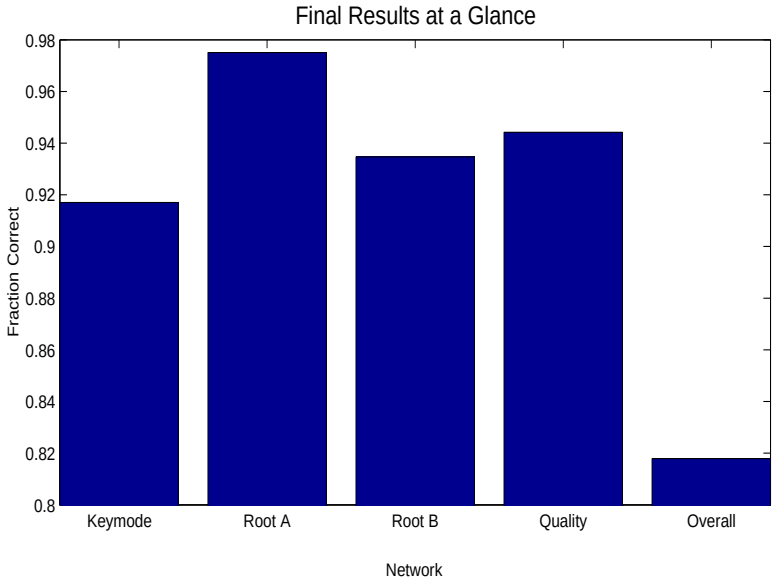

Figure 5.1: Performance of Each Network and Entire System

of *bestcom_quality*. The accuracies of the qualities follow in the exact order of the frequencies with which they occur in the training set.

5.2 Overall Accuracy

Figure 5.1 shows the accuracies of the best results obtained for each Network, as well as the overall accuracy when the Networks are linked serially and tested from beginning to end. The accuracy for the overall results is for the sonorities which had the key, mode, root, and quality all correct. This is a reasonable number considering that the results are obtained serially through four different Networks. All four Networks obtained accuracies over 90%.

5.3 Analysis of Music from Another Composer

It is interesting to see what would happen if music from another composer were fed into the networks trained on Bach chorales. The piece of music selected for testing is Mozart's Piano Sonata in C major, K545 (1st movement). The first 11 bars of this music was fed into the Key Network with window size/position of 8p8n(FF), 120 hidden units, and a voice weight scheme of none. The accuracy obtained was 43.8%. This result was not unexpected. There are several reasons why a low score was achieved. First, the texture of music is different. In Bach chorales, there are four notes in every sonority; some notes are repeated. This means that to the network, it expects every sonority to have three or four distinct notes to its input. However, the Mozart example has a simpler texture. Except for very few sonorities, the entire passage is in two parts. To the network, this means that most of the time a sonority only consists of two notes (and sometimes even one if the two notes are identical), which is not what the network is trained with. Second, in the Mozart example, the harmony is more drawn out. This means that the harmonic function stays the same longer. The harmony in Bach chorales changes rather quickly. Not so in this Mozart example. When the network is fed with music with its harmony more drawn out, it does not adapt to the music; rather, it is ready to change keys at the first hint of a foreign key. In the passage, there is an entire bar of running sixteenth notes on the D minor scale. The correct analysis of this bar is C major; the whole bar should be labeled as II in C major. However, the network thinks the piece has gone to

D minor.

Can the network be re-trained to score higher in this Mozart example? The answer is yes. The network can be re-trained on Mozart piano sonatas. However, some changes would be necessary in order for the re-trained network to perform as well as the network trained on Bach chorales. One change would be to increase the number of hidden units. Amongst themselves, Mozart piano sonatas have a greater variation in style than Bach chorales. Since style is a vague term, consider this thought experiment. The experimenter plays five Bach chorales to a subject (the subject does not know that they are Bach chorales). Then, the experimenter plays a sixth Bach chorale to the subject (again, the type of music is not known to the subject). The subject is asked to rate on a scale the similarness of the sixth piece to the first five. This experiment is repeated on another subject, but this time with Mozart piano sonatas. After testing on many different subjects, the result should be that the score for the Bach chorales is higher. Since the network has to generalize from less similar things, more hidden units would be necessary. Another change would be to increase the window size. Since the harmony is more drawn out, the network needs a greater context to understand properly what is going on in the piece. A third change would be to devise a way to combine cleverly several sonorities into one, so as to reduce the size of the input of the network. This was tried on the Bach chorales (recall the scheme labeled QRTMERGE), but this particular scheme did not work well. However, this scheme, or some variation on it, may work well on Mozart piano sonatas.

5.4 Comparison with Other Algorithms

Krumhansl's key-finding program [13] formed the basis of another key-finding program found on the Internet [2]. There are several differences between [13] and [2]. First, the values of the pre-defined vectors for keys are different. Second, distinction is made between enharmonically equivalent notes. Third, in forming the input vector, instead of summing the total duration of each note in the current segment, a value of 1 is assigned to the corresponding element if the note is present; otherwise, a value of 0 is assigned. Fourth, there is a way of handling modulation.

This modified algorithm's [2] performance was directly compared with *bestcom_key*. After changing a few of the provided user parameters, the best result obtained was a mere 75.1% on the same test suite of Bach chorales. (Recall that *bestcom_key* reached 91.7%.) While not all parameters were changed and tested, it is not immediately clear how they can be changed to improve the accuracy by a considerably large margin. Even if it is possible to obtain over 90% in accuracy, the time taken to find the right set of parameters to achieve this accuracy would make this algorithm not user-friendly.

Taube's work [20] performs tonal analysis, which is slightly different from harmonic analysis. However, its root-finding and quality-finding parts are comparable to the Root Network B and Quality Network of this thesis. On the same set of Bach chorales, Taube's root-finding and quality-finding algorithm performed at 98.2%. Note that as Taube's output was provided statically [1], the two parts of root-finding and quality-finding could not be analyzed separately. The algorithm greatly utilizes lookup tables. By the information given in [20], and also by making some reasonable assumptions

where information is missing, it is shown (in Appendix B) that one Root Network B combined with one Quality Network use only 74% of the space taken up by the lookup tables. Hence, by giving up a few percent of accuracy, significant storage is saved.

5.5 Summary

This Chapter has first presented an analysis of incorrect predictions made by each of the four Networks. Then, it has stated the overall accuracy of the entire network. After this, an analysis of another musical passage from another composer was presented. Finally, comparison was made with other published results.

Chapter 6

Conclusion

6.1 Contributions

This thesis has pushed ahead the state of the art of computerized harmonic analysis. First of all, by using neural networks, it is a novel way of performing harmonic analysis. Previous attempts relied on rule-based methods. Second, the overall performance obtained in this thesis is better than other published results that could be found. In particular, the Key Network performed 16% better than the results found in [2].

This thesis has several practical applications; they will be presented in Section 6.2. One of the pleasant surprises found in this thesis is that the voice weight scheme 'sb' does better than all other voice weight schemes. This is in conformance with the style of Bach chorales which places emphasis on the outer voices. Another discovery was the numerous times the networks with the window size/position of $9p6n$(FF) made it into the best committees (i.e. the committees with the highest accuracies), even though these networks by

themselves did not achieve relatively high accuracies.

6.2 Application of this Work

This thesis has successfully demonstrated the use of neural networks in performing harmonic analysis. This work has potential educational value. It can be used to generate automatically analysis of many musical pieces; music theory students can submit their own analysis online, and they can get immediate feedback as to how their own analysis compared with the correct analysis. Harmonic analysis is a tedious task; from the point of view of a music theory instructor, it is advantageous to have the analysis done automatically. Even if minor corrections to the automatic analysis were to be made, there is still a saving of time and energy. Having such a network online is also an advantage to a music theory student who lives nowhere close to a music theory instructor. In this case, the network is the instructor. In addition to providing a direct comparison, it can be made to provide intelligent feedback on the types of mistakes the student commonly makes. For example, the network might inform the student that "you usually modulate too soon," or "you often mislabel a II chord as a IV chord." Although it might be a little impersonal, at least the student has access to an expert of some kind.

6.3 Future Work

This thesis has focussed on a particular set of pieces of a particular composer. The networks can be trained to perform harmonic analysis on a wide range of composers and styles. In order to perform well on a broader set of pieces, careful selection of the training set is necessary. To be sure, if one were to spend time in expanding the networks to include more composers, one might learn many important truths about what style is.

A measure of style can be carried out by the following experiment. N networks are trained for harmonic analysis, each on pieces of a distinct composer. Then, test all N networks with previous unseen pieces of the same composers. It is reasonable to assume that test pieces of composer A should do the best on the network trained on composer A. However, which of the other networks achieve the next highest score? Assume it is composer B's. This may indicate that the styles of composers A and B are similar. An experiment somewhat similar to this one was described in [10].

Another interesting extension to this thesis is to incorporate the key-detecting ability found in this thesis into a computer accompanist. An accompanist follows a soloist, and usually the accompanist knows what music to play beforehand. All the accompanist has to do is to keep up with the possibly varying speed of the soloist. However, if the soloist is improvising (i.e. making up and playing music on the spot), the accompanist has no score and has to rely on his/her own ear and playing ability to follow along. If a computer were to imitate such an accompanist, the Key Network in this thesis can act as a key-detecting agent, while a second set of neural networks can decide a suitable accompaniment to play, and a third module can deter-

mine the rhythmic information. One major obstacle to overcome is speed, since simulating neural networks is quite slow compared to the speed needed for the computer accompanist to output in real-time.

A final future application of this thesis is as a way of indexing musical databases. A program to index musical databases, called the MUSART Thematic Extractor, is found in [18]; an application that uses this program appeared recently [11]. Potentially, the Key Network of this thesis can determine the key(s) that each piece of music in a certain set has visited. Then, using the key information as index, a user can find out, for example, which pieces had visited C major and G major, but not C minor. This type of information may be useful to investigate into a possible key preference of a composer. Also, the key information can be used in conjunction with other types of information, such as melodic information [18], to aid a user to search for a particular piece of music in a large database.

Bibliography

[1] http://www-camil.music.uiuc.edu/faculty/taube/mtw/mtw.html (accessed August 19, 2002).

[2] http://www.link.cs.cmu.edu/music-analysis (accessed August 19, 2002).

[3] http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/-nnet.shtml (accessed August 17, 2002).

[4] J.S. Bach. 389 chorales. Kalmus Edition K06002, Belwin Mills Publishing Corp.

[5] Jamshed J. Bharucha. Music Cognition and Perceptual Facilitation: A Connectionist Framework. *Music Perception*, 5(1):1–30, 1987.

[6] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, 1995.

[7] Maureen Caudill. Neural Networks Primer Part I. *AI Expert*, pages 46–52, Dec 1987.

[8] R. F. Goldman. *Harmony in Western Music*. WW Norton & Co., New York, 1965.

[9] Hermann Hild, Johannes Feulner, and Wolfram Menzel. HARMONET: A Neural Net for Harmonizing Chorales in the Style of J.S. Bach. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing 4*. Morgan Kaufmann, San Mateo, CA, 1991.

[10] Dominik Hörnel and Wolfram Menzel. Learning Musical Structure and Style with Neural Networks. *Computer Music Journal*, 22(4):44–62, 1998.

[11] Ning Hu, Roger B. Dannenberg, and Ann L. Lewis. A Probabilistic Model of Melodic Similarity. In *Proceedings of the 2002 International Computer Music Conference*, pages 471–474, San Francisco, 2002. International Computer Music Association.

[12] Stefan Kostka and Dorothy Payne. *Tonal Harmony with an Introduction to Twentieth-Century Music*. McGraw-Hill, Toronto, fourth edition, 2000.

[13] Carol L. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, New York, 1990.

[14] Marc Leman. The Ontogenesis of Tonal Semantics: Results of a Computer Study. In P. Todd and G. Loy, editors, *Music and Connectionism*, pages 100–127. The MIT Press, Cambridge, MA, 1991.

[15] Fred Lerdahl and Ray Jackendoff. *A Generative Theory of Tonal Music*. The MIT Press, Cambridge, MA, 1993.

[16] Joseph Machlis and Kristine Forney. *The Enjoyment of Music*. W. W. Norton & Company, New York, sixth edition, 1990.

[17] H. John Maxwell. An Expert System for Harmonic Analysis of Tonal Music. In Mira Balaban, Kemal Ebcioğlu, and Otto Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*. The MIT Press, 1992.

[18] C. Meek and W. P. Birmingham. Thematic Extractor. In *2nd Annual International Symposium on Music Information Retrieval*, pages 119–128, Bloomington, 2001. Indiana University.

[19] Don L. Scarborough, Ben O. Miller, and Jacqueline A. Jones. Connectionist Models for Tonal Analysis. *Computer Music Journal*, 13(3):49–55, 1989.

[20] Heinrich Taube. Automatic Tonal Analysis: Toward the Implementation of a Music Theory Workbench. *Computer Music Journal*, 23(4):18–32, 1999.

[21] David Temperley. An Algorithm for Harmonic Analysis. *Music Perception*, 15(1):31–68, 1997.

[22] Terry Winograd. Linguistics and the Computer Analysis of Tonal Harmony. *Journal of Music Theory*, 2:2–49, 1968.

Appendix A

Performance of Voice Weight Schemes in Various Window Sizes/Positions (of Key Network)

This Appendix presents the graphs for the performance of Key Network for every voice weight scheme of every window size/position attempted. Each graph gives the performance of every voice weight scheme for a particular window size/position.

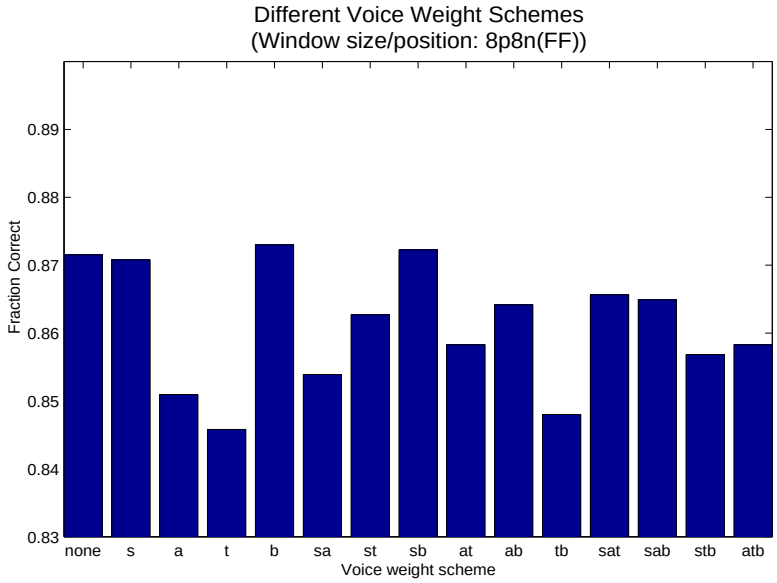

Figure A.1: Performance of Each Voice Weight Scheme in $8p8n$(FF) Networks

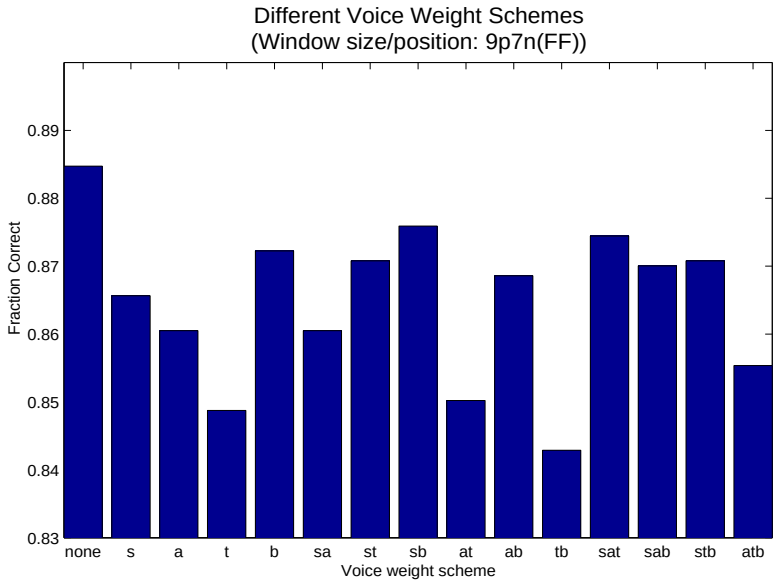

Figure A.2: Performance of Each Voice Weight Scheme in $9p7n$(FF) Networks

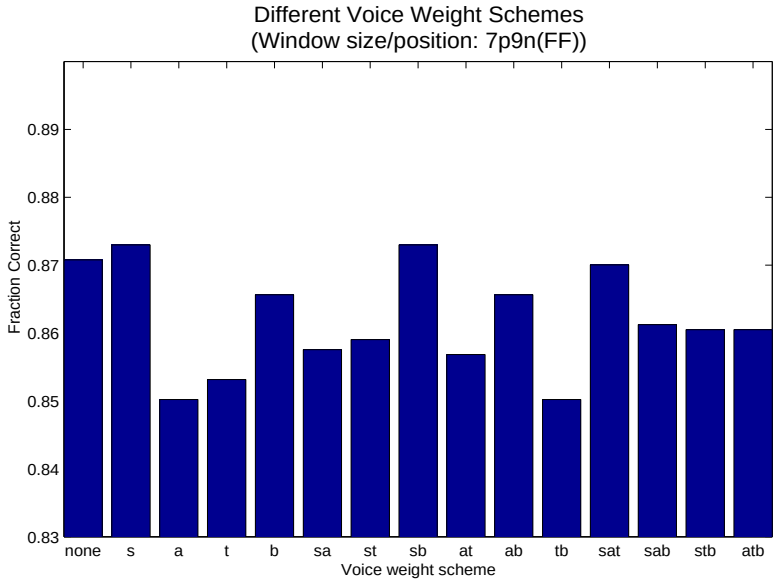

Figure A.3: Performance of Each Voice Weight Scheme in $7p9n$(FF) Networks

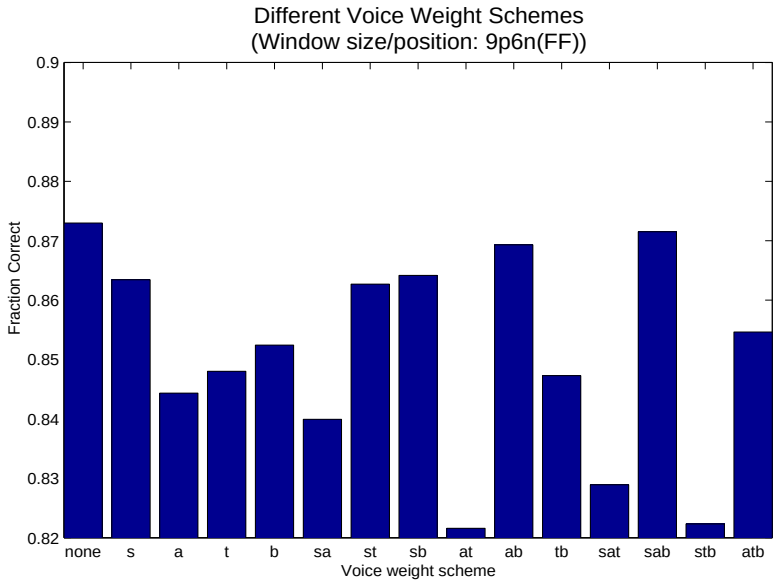

Figure A.4: Performance of Each Voice Weight Scheme in $9p6n$(FF) Networks

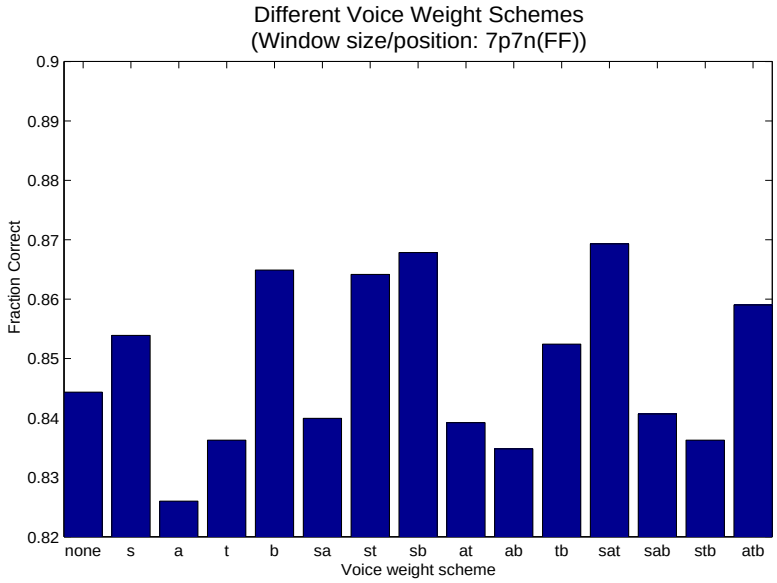

Figure A.5: Performance of Each Voice Weight Scheme in $7p7n$(FF) Networks

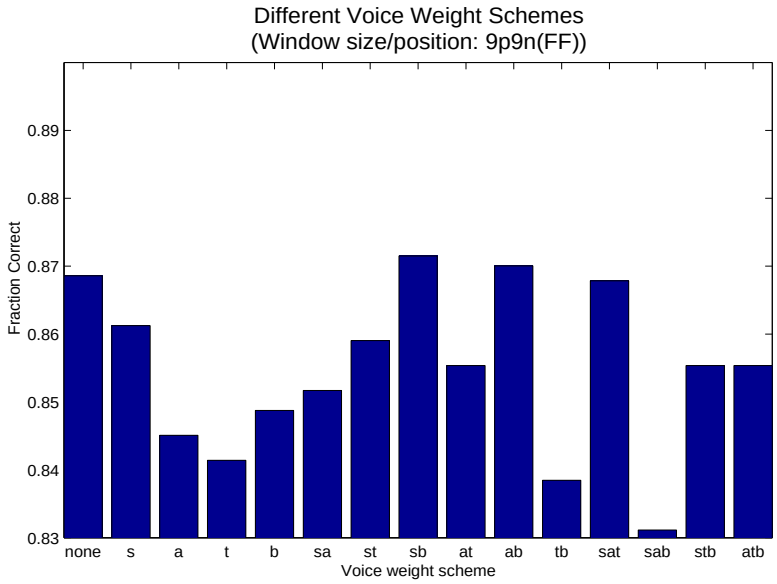

Figure A.6: Performance of Each Voice Weight Scheme in $9p9n$(FF) Networks

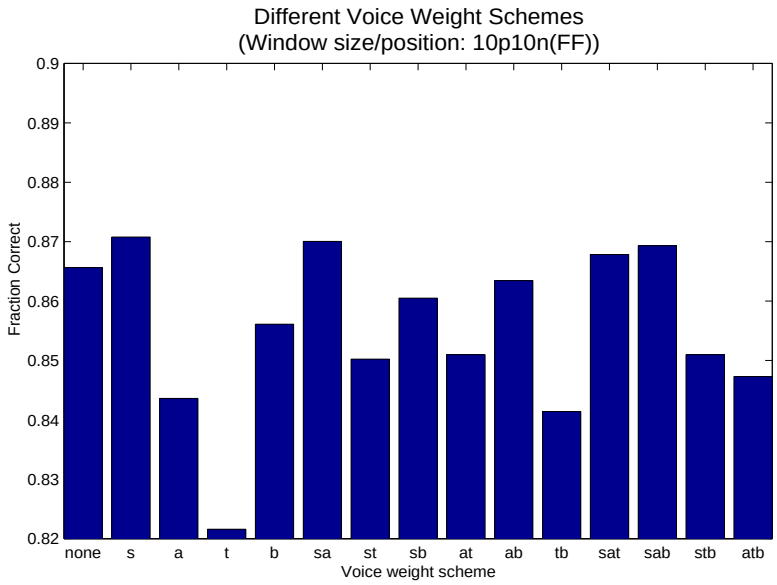

Figure A.7: Performance of Each Voice Weight Scheme in $10p10n$(FF) Networks

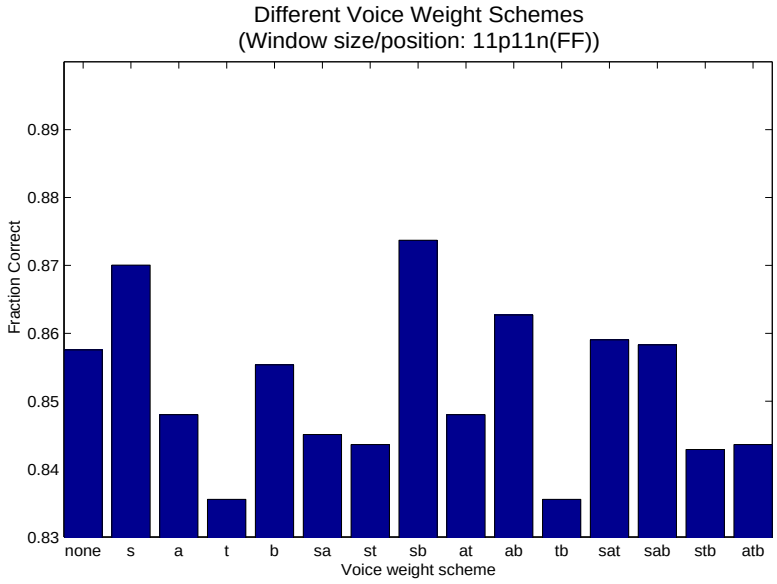

Figure A.8: Performance of Each Voice Weight Scheme in $11p11n$(FF) Networks

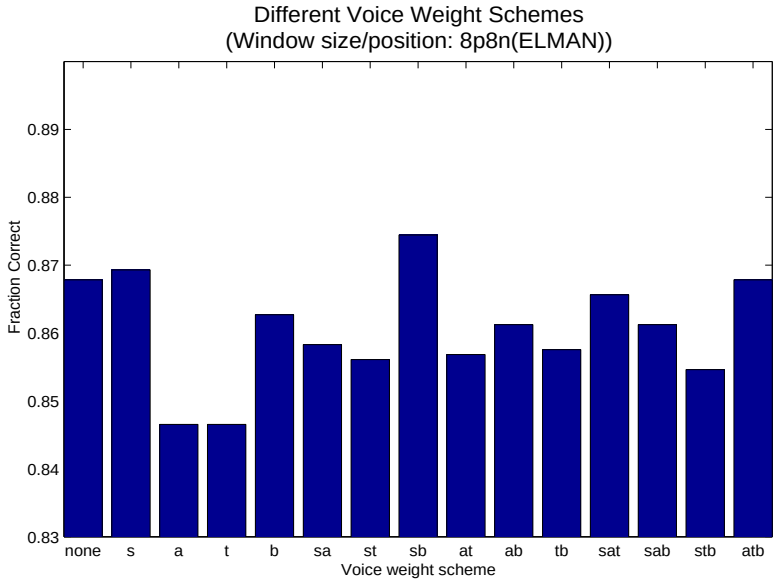

Figure A.9: Performance of Each Voice Weight Scheme in $8p8n$(EL) Networks

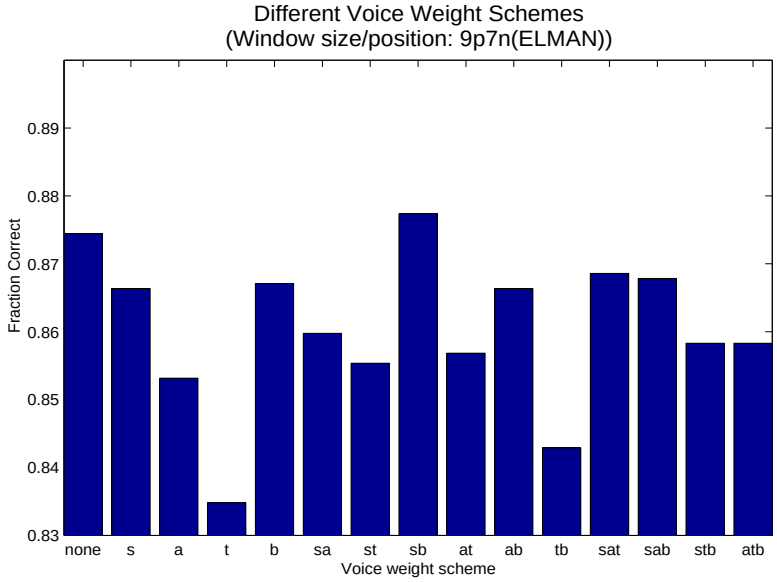

Figure A.10: Performance of Each Voice Weight Scheme in $9p7n$(EL) Networks

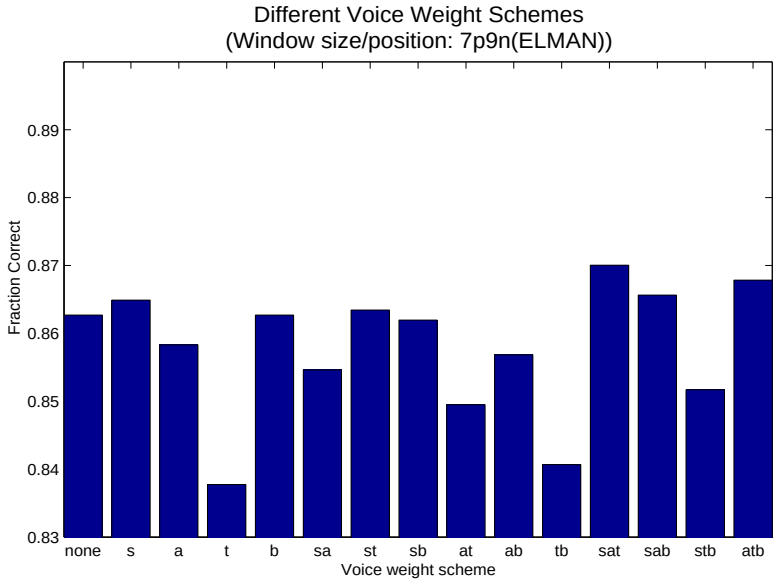

Figure A.11: Performance of Each Voice Weight Scheme in $7p9n$(EL) Networks

Appendix B

Derivations

B.1 Table Size of Taube's Algorithm

An overview of Taube's algorithm is found in Section 2.3. There are two tables involved. The first table takes notes of the sonority and maps them into root letter and quality. For example, the sonority {C, E, G, B} is mapped to {C, +7}. Since the paper [20] states all intervals from doubly diminished to doubly augmented up to an octave are accounted for, this is a total of 38 intervals, and hence the system accounts for 38 distinct notes/roots. For each root, 19 types of sonorities are accounted for: 9 common qualities (as defined in Chapter 2), four 7th chords with the 5th missing, two triads with the 5th missing, and four 7th chords with the 3rd missing. The number of entries in the first table is therefore $38 \times 19 = 722$. For each entry, the root can be encoded in 6 bits, while the quality can be encoded in 3 bits (the system seems only to differentiate between 3 types of 7th chords, for a total of 7 qualities; this thesis' tells apart 5). The index takes up 6 bits/note $\times$ 4

notes = 24 bits. Each entry therefore uses $6 + 3 + 24 = 33$ bits. This table uses 722 entries$\times$ 33 bits/entry = 23826 bits.

The second table takes the root of the sonority and the current tonal center and maps them into an interval. The table size is $38 \times 38 = 1444$. Each entry takes up 6 bits for the interval and 11 bits for the index, for a total of 17 bits. This table uses $1444 \times 17 = 24548$ bits. The total size of the two tables is 48374 bits.

In contrast, consider Root Network B with 20 hidden units and window size/position $1p1n$. There are 19 notes/sonority $\times$ 3 sonorities = 57 units in the input layer (plus a bias), 20 units in the hidden layer (plus a bias), and 9 units in the output layer. The total number of weights is $58 \times 20 + 21 \times 9 = 1349$. Assume that each weight falls in the range [-1, 1], and the precision is 3 decimal places. Each weight then can be represented in 11 bits (10 bits for the mantissa + 1 sign bit). The size of the network is $1349 \times 11 = 14839$. Quality Network's derivation is identical, except that the input layer contains 9 extra inputs per sonority from Root Network B. The total for Quality Network comes out to 20779. Therefore, the two networks combine for a total of 35618 bits. And $35618/48374 = 0.74$. Granted, given the relatively small sizes, this is just a theoretical comparison.