Multiple Fundamental Frequency Pitch Detection for Real Time MIDI Applications

Nathan Hilbish
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd
Part of the Engineering Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/2853

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.
Multiple Fundamental Frequency Pitch Detection for Real Time MIDI Applications

A Thesis Presented in Partial Fulfillment of the Requirements of the Degree Master of Science at Virginia Commonwealth University

By

Nathan Hilbish
BSc in Electrical Engineering, Virginia Commonwealth University, 2010

Director: Afroditi Vennie Filippas
Associate Dean of Undergraduate Studies, School of Engineering

Virginia Commonwealth University
Richmond, Virginia
August 2012
ACKNOWLEDGMENTS

Foremost I would like to thank my advisor Dr. Afroditi Vennie Filippas for the continuing support of my research, and for all the support and guidance that has helped me through both my undergraduate and graduate career. I could not have asked for a better advisor to help me complete my research.

I would also like to acknowledge my advisory committee, Dr. Alen Docef, and Dr. Kayvan Najarian for their input, support, and insightful help in the process of completing my thesis research.

I would also like to acknowledge Virginia Commonwealth University for supporting my research.

I would like to thank my family and friends for supporting me throughout my educational career.

Last but not least I would like to thank my girlfriend, Ashley Greer, for giving me all her love, support, and patience as I completed my thesis research.
# TABLE OF CONTENTS

List of Tables .................................................................................................................. VII
List of Figures .................................................................................................................. VIII
Abstract ........................................................................................................................... X

CHAPTER 1: Introduction ............................................................................................... 1
1.1 Introduction ............................................................................................................... 1
1.2 Music Background .................................................................................................... 1
1.2.1 Musical Tuning and Notes .................................................................................. 2
1.2.2 Musical Chords and Scales ................................................................................ 3
1.2.2.1 Major and Minor Chords .............................................................................. 4
1.2.2.2 Other Chord Combinations ........................................................................ 5
1.2.2.3 Constructing Chords .................................................................................... 5
1.3 The Guitar Signal ...................................................................................................... 7
1.4 Musical Instrument Digital Interface ....................................................................... 8
1.5 Neural Networks: Technical Background .................................................................. 9
1.5.1 Feed Forward Neural Network Architecture ..................................................... 9
1.5.1.1 Input Layer .................................................................................................. 10
1.5.1.2 Hidden Layers ............................................................................................ 10
1.5.1.3 Output Layer ................................................................................................ 11
1.5.1.4 Bias ............................................................................................................. 12
1.5.1.5 Activation Function ...................................................................................... 13
1.5.1.6 Saturated Linear Activation Function ......................................................... 14
1.5.2 Multi-Class Neural Networks ............................................................................. 15
1.5.2.1 Binary Classification Tree ........................................................................... 15
1.5.3 Data Organization for model training ............................................................... 16
1.5.3.1 Validation .................................................................................................... 16
1.5.3.2 Three-Way Data Splits .............................................................................. 17
1.5.3.3 Cross-Validation ........................................................................................ 18
2.3.4.1 Communication Theory Motivated Error Correction ................................................. 40
2.3.4.2 Repetition Coding ........................................................................................................ 40

CHAPTER 3: Results ..................................................................................................................... 42
3.1 Measuring Performance ....................................................................................................... 42
3.1.1 Accuracy and Precision .................................................................................................... 42
3.1.2 Recall (Sensitivity) and Specificity ................................................................................ 43
3.1.3 Confusion Matrix ............................................................................................................ 43
3.2 Results of Root Note Detection .......................................................................................... 44
3.3 Polyphony Estimation Results ............................................................................................. 48
3.4 Chord Recognition Results ................................................................................................. 51
3.5 Comparison to Other Techniques ...................................................................................... 56
3.6 Benchmarks on Embedded Platform .................................................................................. 57

CHAPTER 4: Discussion .............................................................................................................. 59
4.1 Root-Note Detection ........................................................................................................... 59
4.2 Polyphony Estimation ........................................................................................................ 59
4.3 Chord Recognition and Error Correction .......................................................................... 60
4.4 Comparing Algorithms ....................................................................................................... 60
4.5 Real-Time performance ....................................................................................................... 61

CHAPTER 5: Conclusion and Future Work .................................................................................. 62
5.1 Conclusions ....................................................................................................................... 62
5.2 Future Work ...................................................................................................................... 62

Bibliography ............................................................................................................................... 64
Curriculum Vitae ........................................................................................................................ 67
LIST OF TABLES

Table 1.1: List of musical intervals with corresponding frequency ratios ................................................................. 3
Table 1.2: List of chords used for classifier training and evaluation ........................................................................ 6
Table 3.1: Summary of performance statistics for root note selection classifier ..................................................... 48
Table 3.2: Summary of performance statistics for polyphony estimation ................................................................. 51
Table 3.3: Summary of performance statistics for chord recognition for polyphony known a priori and estimated
polyphony ................................................................................................................................................................... 55
Table 3.4: Summary of performance statistics for chord recognition for polyphony known a priori and estimated
polyphony ................................................................................................................................................................... 56
Table 3.5: Execution times each function of the proposed algorithm ........................................................................ 57
LIST OF FIGURES

Figure 1.1: Transient response of a guitar string that has been physically displaced .............................................. 7
Figure 1.2: Frequency spectrum of a typical guitar note ................................................................. 8
Figure 1.3: Distribution of frequencies across the MIDI note range .................................................... 9
Figure 1.4: General Feed Forward Artificial Neural Network Architecture ........................................... 10
Figure 1.5: Effect of bias weight value on activation function ......................................................... 13
Figure 1.6: Comparison of activation functions: sigmoid (blue), bi-polar sigmoid (green), and hyperbolic tangent (red) .......................................................................................................................... 14
Figure 1.7 Symmetric binary tree with four classes (top) and skewed binary tree with three classes (bottom) .. 16
Figure 1.8: Block diagram of Klapuri’s multiple fundamental frequency estimation system .................. 24
Figure 1.9: Block diagram from the TK method for multiple fundamental frequency pitch detection ........ 25
Figure 2.1: Multiple fundamental frequency algorithm overview ................................................... 27
Figure 2.2: Frequency Spectrum of total frequency range of FFT output (blue) and reduced frequency output (red)
for upper bound of target F0 frequencies (~517 Hz).................................................................................. 29
Figure 2.3: Frequency Spectrum of total frequency range of FFT output (blue) and reduced frequency output (red)
for lower bound of target F0 frequencies (~86 Hz)................................................................................. 30
Figure 2.4: Magnitude spectrum after normalizing between -1 and 1 ......................................................... 31
Figure 2.5: Block Diagram of multiple-F0 classification algorithm ..................................................... 32
Figure 2.6: Data flow example of root note selection binary tree classifier ............................................ 34
Figure 2.7: Block diagram view of each feed-forward network in binary decision tree from root note recognition35
Figure 2.8: Block diagram view of each feed-forward network in binary decision tree from root note recognition36
Figure 2.9: Skewed binary classification tree showing an example of data flow when a two-note chord is presented at the input .................................................................................................................. 37
Figure 2.10: Grouping of notes in a binary tree based on musical relationships ........................................ 39
Figure 2.11: Block diagram of basic communication system ............................................................ 40
Figure 3.1: Comparison of example confusion matrix table and plot ................................................... 44
Figure 3.2: Confusion matrix for root selection classifier .................................................................... 45
Figure 3.3: Histogram of precision values for root selection classifier ................................................................. 46
Figure 3.4: Histogram of recall (sensitivity) values for root selection classifier ...................................................... 46
Figure 3.5: Histogram of specificity values for root selection classifier ................................................................. 47
Figure 3.6: Histogram of error rates for root selection classifier .............................................................................. 47
Figure 3.7: Confusion matrix for polyphony estimation ............................................................................................ 49
Figure 3.8: Plot of precision values for polyphony estimation .................................................................................. 50
Figure 3.9: Plot of recall (sensitivity) values for polyphony estimation ................................................................. 50
Figure 3.10: Plot of specificity values for polyphony estimation .............................................................................. 50
Figure 3.11: Plot of error rate values for polyphony estimation ............................................................................... 51
Figure 3.12: Confusion matrices for chord recognition when polyphony is known a priori (left) and is estimated
(right) ................................................................................................................................................................. 52
Figure 3.13: Histogram of precision values for chord recognition; Polyphony known a priori (red grey), polyphony
estimated (blue) .................................................................................................................................................. 53
Figure 3.14: Histogram of recall (sensitivity) values for chord recognition; Polyphony known a priori (red),
polyphony estimated (blue) .................................................................................................................................. 53
Figure 3.15: Histogram of specificity values for chord recognition; Polyphony known a priori (red), polyphony
estimated (blue) .................................................................................................................................................. 54
Figure 3.16: Histogram of error rate values for chord recognition; Polyphony known a priori (red), polyphony
estimated (blue) .................................................................................................................................................. 54
Figure 3.17: Confusion matrices for error corrected output when three comparisons are used (left), and five
comparisons are used (right) .................................................................................................................................... 56
Figure 3.18: Comparison of error rates from different algorithms for detecting fundamental frequencies ....... 57
ABSTRACT

MULTIPLE FUNDAMENTAL FREQUENCY DETECTION FOR REAL TIME MIDI APPLICATIONS

Nathan Kyle Hilbish, Master of Science

A Thesis Presented in Partial Fulfillment of the Requirements of the Degree Master of Science at Virginia Commonwealth University

Virginia Commonwealth University, 2012

Major Director: Afroditi Vennie Filippas
Associate Dean of Undergraduate Studies, School of Engineering

This study aimed to develop a real time multiple fundamental frequency detection algorithm for real time pitch to MIDI conversion applications. The algorithm described here uses neural network classifiers to make classifications in order to define a chord pattern (combination of multiple fundamental frequencies). The first classification uses a binary decision tree that determines the root note (first note) in a combination of notes; this is achieved through a neural network binary classifier. For each leaf of the binary tree, each classifier determines the frequency group of the root note (low or high frequency) until only two frequencies are left to choose from. The second classifier determines the amount of polyphony, or number of notes played. This classifier is designed in the same fashion as the first, using a binary tree made up of neural network classifiers. The third classifier classifies the chord pattern that has been played. The chord classifier is chosen based on the root note and amount of polyphony, the first two classifiers constrain the third classifier to chords containing only a specific root note and a set polyphony. This allows for the classifier to be more focused and of a higher accuracy. To further increase accuracy, an error correction scheme was devised based on repetitive coding, a
technique that holds out multiple frames and compares them in order to detect and correct errors. Repetitive coding significantly increases the classifiers accuracy; it was found that holding out three frames was suitable for real-time operation in terms of throughput, though holding out more frames further increases accuracy it was not suitable real time operation. The algorithm was tested on a common embedded platform, which through benchmarking showed the algorithm was well suited for real time operation.
CHAPTER 1: INTRODUCTION

1.1 Introduction

Pitch detection is a means of extracting frequency information from an acoustic signal and is an important research topic in signal processing. It is generally considered a solved problem for monophonic (single voice) recordings, but remains a non-trivial, unsolved problem in a real-time setting where accurate pitch detection must be achieved with stringent time constraints (de la Cuadra, Master, & Sapp, 2001), (Klapuri A., 2004). For polyphonic signals, that is, signals containing multiple voices or notes, the complexity of pitch detection increases dramatically. Even when analyzing prerecorded signals for transcription applications, there is much room for improvement on current techniques (Thomas, 2012). Many techniques trade off speed performance for accuracy, while others do the opposite. This research aims to develop an algorithm to detect multiple fundamental frequencies of a guitar signal and convert these detected frequencies into MIDI control signals while operating with high accuracy in a real-time environment with music performance in mind.

1.2 Music Background

Music theory can provide several meaningful ways of separating data based on how notes and chords are classified within the realm of western music theory. Though much of music theory is based on how musical signals are perceived, it also has a mathematical basis which may be useful to separate and organize music note data so that a classification can be made regarding what notes make up a multi-pitched sound and how many notes make up that sound.
1.2.1 Tuning and Notes

Guitars are tuned based on an equal temperament system of tuning where each octave of notes is divided into equal steps of twelve notes or semitones (Rossing, Wheeler, & Moore, 2002). These notes are C, C#, D, D#, E, F, F#, G, G#, A, A#, and B. These notes are repeated as octaves, for example the note A occurs at frequencies 55 Hz, 110 Hz, 220 Hz, 440 Hz, and so on. Other note frequencies can be calculated using A-440Hz as a reference and using equation 1.1, where \( f \) is the frequency of the note and \( d \) is a relative note number where \( d=69 \) is frequency A-440Hz.

\[
f = 2^{(d-69)/12} \cdot 440
\]  

(1.1)

Notes are given names that correspond to their relationship with regard to how many semitones follow and precede them. This relationship can be measured in cents, where a cent is a logarithmic unit of measurement where the distance between each semitone is 100 cents (Rossing, Wheeler, & Moore, 2002). The size of the cent interval can be calculated using equation 1.2, where \( f_1 \) is a frequency corresponding to a note and \( f_2 \) is the frequency of a note whose distance from \( f_1 \) \( n \) measured in cents. Table 1.1 shows the relationship between musical interval names, the frequency ratio, and cents measurement.

\[
n = 1200 \cdot \log_2 \left( \frac{f_2}{f_1} \right)
\]  

(1.2)
Table 1.1: List of musical intervals with corresponding frequency ratios

<table>
<thead>
<tr>
<th>Number of Semitones</th>
<th>Musical interval</th>
<th>Interval Number</th>
<th>Frequency Ratio</th>
<th>Cents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Perfect Unison</td>
<td>P1</td>
<td>1.0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Minor Second</td>
<td>m2</td>
<td>1.0595</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Major Second</td>
<td>M2</td>
<td>1.1225</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Minor Third</td>
<td>m3</td>
<td>1.1892</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>Major Third</td>
<td>M3</td>
<td>1.2599</td>
<td>400</td>
</tr>
<tr>
<td>5</td>
<td>Perfect Forth</td>
<td>P4</td>
<td>1.3348</td>
<td>500</td>
</tr>
<tr>
<td>6</td>
<td>Augmented Fourth</td>
<td>A4</td>
<td>1.4142</td>
<td>600</td>
</tr>
<tr>
<td>7</td>
<td>Perfect Fifth</td>
<td>P5</td>
<td>1.4983</td>
<td>700</td>
</tr>
<tr>
<td>8</td>
<td>Minor Sixth</td>
<td>m6</td>
<td>1.5874</td>
<td>800</td>
</tr>
<tr>
<td>9</td>
<td>Major Sixth</td>
<td>M6</td>
<td>1.6818</td>
<td>900</td>
</tr>
<tr>
<td>10</td>
<td>Minor Seventh</td>
<td>m7</td>
<td>1.7818</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>Major Seventh</td>
<td>M7</td>
<td>1.8877</td>
<td>1100</td>
</tr>
<tr>
<td>12</td>
<td>Perfect Octave</td>
<td>P8</td>
<td>2.0000</td>
<td>1200</td>
</tr>
</tbody>
</table>

1.2.2 Musical Chords and Scales

Musical notes played on the guitar are commonly grouped based on relationships derived from musical scales. The two most commonly used scales on which chords are built are the major scale and the minor scale (Oudre, Grenier, & Fevotte, 2009). The major scale consists of notes following the pattern of whole steps (two semitones) and half steps (one semitone) as follows: whole – whole – half – whole – whole – whole – half.
The minor scale follows the pattern whole – half – whole – whole – half – whole – whole.

Chords are constructed from these musical intervals to provide consonance or dissonance, where consonance is a musical sound that is considered to be at rest (Benson, 2008) and dissonance occurs when a musical sound is “unstable”. Mathematically speaking, the harmonics in a signal that has consonance align or are spaced far apart as to not conflict sonically. This occurs when harmonics are close to one another, for example playing a root note (unison) with a minor second or major seventh will produce a dissonant sound that is perceived as sounding “tense”. The description of a sound being “tense” can be attributed to frequency beating, where each frequency greatly affects the amplitude of the other. Intervals that produce sounds with “high” consonance or perfect consonance are octaves, perfect fourths, and perfect fifths. This is because these intervals are harmonic ratios of each other, with an octave being twice the frequency of the root note, and a perfect fifth having a ratio of approximately 1.5. Due to the harmonic relationship between the root note and the perfect fifth, the perfect fifth interval is almost always included in a chord. Intervals producing “low” consonance or imperfect consonance are major and minor thirds, as well as major and minor sixths. These produce tones that are far enough away from the root note tones to produce a “stable” sound.

1.2.2.1 Major and Minor Chords

Major chords are constructed using the intervals found in the major scale. The major chord consists of three notes: the root note, major third, and perfect fifth. By adding intervals such as octaves, sevenths, ninths, and thirteenths, a musician can add consonance or dissonance. It has been found through prior research in (Oudre, Grenier, & Fevotte, 2009) that the majority of chords played in popular music consist of major chords.
The second most common chords played are minor chords; these chords are constructed in the same fashion as major chords except a major third interval is replaced by a minor third interval.

1.2.2.2 Other Chord Combinations

Other less common chord combinations include dominant, suspended, augmented, and diminished chords (Hal Leonard Corporation, 2004). Dominant chords are major chords with an added minor seventh interval. Suspended chords can be of two types: suspended fourth and suspended second; suspended fourth chords are neither major nor minor and have intervals that include a perfect fifth, and perfect forth. The suspended second chord is similar to a suspended fourth chord but has intervals that include a perfect fifth and major second. These chords are often used to transition between major and minor chords. The final chord types discussed here are augmented and diminished chords. Augmented chords are major chords using a perfect fifth interval that has been sharpened or raised in frequency by one semitone. Diminished chords are the opposite of augmented chords in that the perfect fifth interval has been flattened by one semitone.

1.2.2.3 Constructing Chords

Chords constructed for this research can be seen in Table 1.2 with each chord’s corresponding construction formula. As polyphony is increased, higher octave intervals are added first, and intervals corresponding to octaves plus a perfect fifth are added. For example, a chord having a polyphony of six would include the root note, a perfect fifth, an octave interval (since this is the next interval in terms of frequency) a major third, an octave plus a perfect fifth, and finally an interval three octaves higher than the root note.
<table>
<thead>
<tr>
<th>Chord</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major</td>
<td>P1+M3+P5</td>
</tr>
<tr>
<td>Major 7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+M3+P5+M7</td>
</tr>
<tr>
<td>Major 9&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+M3+P5+M7+M9</td>
</tr>
<tr>
<td>Major 13&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+M3+P5+M7+M9+M13</td>
</tr>
<tr>
<td>Minor</td>
<td>P1+m3+P5</td>
</tr>
<tr>
<td>Minor 7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+m3+P5+m7</td>
</tr>
<tr>
<td>Minor 9&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+m3+P5+m7+m9</td>
</tr>
<tr>
<td>Minor 13&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1+m3+5+m7+m9+m13</td>
</tr>
<tr>
<td>Dominant 6</td>
<td>P1+M3+P5+M6</td>
</tr>
<tr>
<td>Dominant 7</td>
<td>P1+M3+P5+m7</td>
</tr>
<tr>
<td>Dominant 9</td>
<td>P1+M3+P5+m7+M9</td>
</tr>
<tr>
<td>Dominant 13</td>
<td>P1+M3+P5+m7+M9+M13</td>
</tr>
<tr>
<td>Suspended 4&lt;sup&gt;th&lt;/sup&gt;</td>
<td>P1-P4-P5</td>
</tr>
<tr>
<td>Suspended 2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>P1-M2-P5</td>
</tr>
<tr>
<td>Augmented</td>
<td>P1-M3-#P5</td>
</tr>
<tr>
<td>Diminished</td>
<td>P1-M3-bP5</td>
</tr>
</tbody>
</table>

Table 1.2: List of chords used for classifier training and evaluation
1.3 The Guitar Signal

Guitar signals have several physical attributes that are important to keep in mind when designing a pitch detector for multiple fundamental frequencies. Like most acoustic signals generated by musical instruments, guitar signals follow an amplitude envelope (Rossing, Wheeler, & Moore, 2002) where the signal builds up in amplitude when a string is physically displaced, then decays once the initial transient from the initial displacement has passed. Guitar signals have a very fast attack portion of the signal, with a slow decay compared to the attack as seen in Figure 1.1.

![Figure 1.1: Transient response of a guitar string that has been physically displaced](image)

Another important attribute of a guitar signal is the frequency response and how harmonics are related to each other. It can be seen in Figure 1.2 that the fundamental frequency has a smaller magnitude than higher order partials, up to the seventh harmonic. This explains why simple methods based on autocorrelations or peak-picking often misidentify the fundamental frequency of a guitar signal (de la Cuadra, Master, & Sapp, 2001).
1.4 Musical Instrument Digital Interface

Musical Instrument Digital Interface (MIDI) is a standard communication protocol in which digital musical instruments and other devices can communicate. There are a multitude of possible commands that can be sent through MIDI messages, but for this research, the focus will be on pitch messages; for further detail on other MIDI messages please refer to MIDI 1.0 Specifications Manual (MIDI Manufactures Association, 1995). MIDI signals have a range of 128 notes ranging in frequency from 8.17Hz to 12.5Hz. These MIDI notes are a linear mapping of the logarithmic relationship between pitch and frequency described by equation 1.3, where $m$ is a MIDI note number and $f_m$ is the corresponding MIDI frequency. Figure 1.3 shows the distribution of frequencies across the MIDI note range, with the typical range of a guitar highlighted in red.

$$m = 12\log_2\left(\frac{f_m}{440}\right)$$  \hspace{1cm} (1.3)
1.5 Neural Networks: Technical Background

Artificial neural networks (ANN) are a class of algorithms that can learn a variety of functions from taught examples. ANN learns through supervised learning; this is where the network is presented with example training data that corresponds to known outputs. Neural networks are well suited for classification applications where an algorithm is needed to define specific classes an input belongs to.

1.5.1 Feed Forward Neural Network Architecture

A Feed Forward Neural network is a structure of fully interconnected layers as seen in Figure 1.4. This network structure is referred to as feed forward because the data only flows in the forward direction between each layer (Parker, 2006). Each layer contains any number of processing units called “neurons”; each neuron is fully interconnected to neurons in adjacent layers through weight values. These weights determine the data flow through the network and are adjusted during the training phase to produce correct outputs. Weights are used to store
learned information. Lastly each node in the hidden layer and output layer has a processing function associated with it, known as an activation function; each activation function serves to map the summation of the weighted inputs from the previous layer to a range defined by the activation function.

Figure 1.4: General Feed Forward Artificial Neural Network Architecture

1.5.1.1 Input Layer

The input layer provides interconnection between input data and the ANN Data. The neurons in the input layer typically provide no processing (Parker, 2006). Each output from the input layer is labeled $x_j$, where $j$ has a range of 1 to $d$ number of input neurons.

1.5.1.2 Hidden Layers

There can be any number of hidden layers in an ANN; for most problems, one hidden layer with an appropriate number of neurons will perform well, but for more complex problems, two layers
may be used (Parker, 2006). As a rule of thumb, a network with no hidden layers (perceptron) can represent any linearly separable function, a network with one hidden layer can represent a function that contains continuous mapping from one finite space to another, while a network with two hidden layers can approximate any smooth mapping to any accuracy given an appropriate amount of neurons (Heaton, 2008). For illustration purposes from this point forward, we will assume a network only has one hidden layer. Each value $x_j$ coming from the input layer has a weight associated with it that connects to each neuron in the hidden layer. Each hidden neuron $z_h$, where $h$ represents hidden layer neurons 1 to $H$, is interconnected via the weight $w_{hj}$.

The sum of the inputs to the hidden layer, $z_h$ is calculated in equation 1.4.

$$z_h = \sum_{j=0}^{d} w_{hj} x_j$$  \hspace{1cm} (1.4)$$

The activation function is applied to $z_h$ giving $a_h$ shown in equation 1.5. A typical activation function used is the hyperbolic tangent discussed further in 1.5.1.5.

$$a_h = \tanh(z_h)$$  \hspace{1cm} (1.5)$$

1.5.1.3 Output Layer

The output layer involves a computation similar to that of the hidden layer, where the outputs of the hidden layer $a_h$ are summed together as seen in equation 1.6, and $v_{ih}$ are the weight values connecting the hidden layer to the output layer with $i$ representing the number of output neurons and $o_i$ the output values.
The outputs $o_i$ can be processed by an activation function as was done in the hidden layer, this activation function may be linear or sigmoid, like the hyperbolic tangent. For classification problems it is often useful to map the output to hard values such as zero or one; to this end, a rounding function may be used to map to the nearest integer value.

1.5.1.4 Bias

Bias terms are special neurons added to the terms $x_j$ and $a_h$. Bias values are always assigned a value of one and their associated weight is adjusted via training just as other weight values are. These neurons are included to increase the network’s capacity to solve problems by allowing the hyper-planes that separate classes to be offset for better positioning (Leverington, 2009). Mathematically, it serves to shift the activation function left or right as seen in Figure 1.5.
1.5.1.5 Activation Function

As stated in 1.5.1, activation functions serve as processing units for each neuron. A non-linear differentiable monotonically increasing function such as a sigmoid function, a bi-polar sigmoid function, or a hyperbolic tangent function, whose transfers are defined in equations 1.7, 1.8, and 1.9, is typically used, and can be viewed in Figure 1.6. It has been shown in (Karlik & Olgac, 2010) that the hyperbolic tangent provides the best performance in regards to training speed and accuracy of classification.

\[
\text{sig}(x) = \frac{1}{1 + e^{-x}} \tag{1.7}
\]

\[
\text{bisig}(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{1.8}
\]

\[
\text{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.9}
\]
Figure 1.6: Comparison of activation functions: sigmoid (blue), bi-polar sigmoid (green), and hyperbolic tangent (red)

1.5.1.6 Saturated Linear Activation Function

For real-time processing applications, it can be useful to approximate the hyperbolic tangent function with a piecewise linear approximation to increase speed performance. Typically, the hyperbolic tangent function is computed via a lookup table; this can be computationally expensive (Namin, 2009) compared to a linear approximation. A saturated linear function, defined in equation 7, may be used to approximate the hyperbolic tangent function and easily implemented using a simple conditional statement seen below in

$$
satlin(x) = \begin{cases} 
-1, & x \leq -1 \\
 x, & -1 < x < 1 \\
 1, & x \geq 1 
\end{cases}
$$

(7)
1.5.2 Multi-Class Neural Networks

Neural networks classifiers are typically well suited for binary classification problems were the network is designed for classification of two class problems. There are several methods to extend two-class classifiers into multi-class classifiers, including breaking the classification into several two-class problems, which will be the primary method used in this research. For further information on other multi-class schemes refer to (Ou & Murphey, 2007).

1.5.2.1 Binary Classification Tree

A neural network binary classification tree is a system of binary neural networks arranged in a classification tree structure. Each neural network is tasked with only making classifications between two classes. The final class that an input belongs to can be identified by following a path from the root node (first network) to the leaf node (last network that was activated). It can be seen in Figure 1.7 that data flows from root 1-4 to leaf 2. Binary classification trees may be symmetric, with \(2^n\) classes, where \(n\) is \(l-1\), where \(l\) is the number of levels in the binary tree.

Algorithm 1.1: Saturated linear activation Function pseudo-code

1.1 if \(i \leq -1\)  
1.2 \(i \leftarrow -1\)  
1.3 else  
1.4 if \(i \geq 1\)  
1.5 \(i \leftarrow 1\)  
1.6 end if  
1.7 else  
1.8 \(x \leftarrow x\)  
1.9 end if
They may also be skewed; any number of classes may exist, and branches will have more or fewer classes (leafs) to classify.

![Symmetric and skewed binary trees]

**Figure 1.7** Symmetric binary tree with four classes (top) and skewed binary tree with three classes (bottom)

1.5.3 Data Organization for model training

For neural networks to learn, data must be presented to them during a training phase in which weights are updated to produce a desired output based on the given input data. The data must be organized in such a way that only part of the data is used for training and the rest for testing; this helps to ensure the network is not over fit; more on over fitting is discussed in 1.5.3.3.1. This section gives an overview of organizing data to properly train a neural network in order to train and design networks accurately and efficiently.

1.5.3.1 Validation
Validation methods are used for two tasks when designing a classifier model, firstly to evaluate the performance of the model, and secondly to choose an appropriate model (Duda, Hart, & Stork, 2001). To estimate the performance of a model, ideally one would have access to a complete range of possible examples of input data, allowing the choice of a model that gives the best overall performance across a complete dataset. In actuality, datasets are typically only partially representative of a whole population. This is why methods must be used to carefully train a network so that given exposure to a small dataset a model can accurately fit data to a desired output with low error.

1.5.3.2 Three-Way Data Splits

Data can be organized into three subsets, training, validation, and testing. Each serves an important function in designing an appropriate model for classification or training as well as providing insight to the performance of the model during training and implementation phases. Training data is used for updating and tuning parameters of a classifier to best fit the desired input-output relationship of the model.

Validation data is the segment of a dataset used to evaluate the classifier during training and is not used to directly tune parameters. In the case of back propagation ANNs, the validation set is used to monitor the performance of the ANN as it is training. As the network is trained, the validation error typically decreases along with the training set error (Beale, Hagan, & Demuth, 2012). When a network becomes overfit to its training data, a rise in the validation error will occur. Most learning algorithms save the state of the network weights before the error increase occurred, which is a model that produced the minimum error for the given data set.
Testing data is not used during the training phase of a classifier. It is useful to leave out a selection of data to evaluate the trained model on a dataset it has not been exposed to as well as to compare different models performance.

1.5.3.3 Cross-Validation

1.5.3.3.1 Over Fitting

It is important in pattern recognition and classification problems to choose a model that is both accurate and flexible. A system may be designed to perfectly classify a complex dataset, but may perform very poorly when presented with a new but similar dataset (Duda, Hart, & Stork, 2001). This is known as over fitting. It is important to have a method for choosing a suitable model as to not be so complex, it is only valid for a small subset of data, but not so simple it fails at classifying the majority of data it is presented.
K-Fold Cross Validation

Simply evenly splitting the dataset into training, validation, and testing sets may lead to overfitting, particularly in sparse datasets (Duda, Hart, & Stork, 2001). One solution to ensure the classifier is not over fit to a subset of data is to divide the data into $k$ subsets leaving out a set for validation and train the classifier $k$ times each time leaving out a set for validation. If the performance of each of the $k$ classifiers is similar then the model chosen is shown to perform with the same performance on a wide range of data.

K-fold cross validation is also a useful tool for choosing the best model for final implementation of the classifier. Typically, out of the $k$ classifiers trained, there will be a classifier with a higher performance than others, but if the model is well fit to the data then each classifier should have similar performances. This leads to a way not only to evaluate the model is appropriate for the task at hand but also allows for choice of the best model for the job.

10-Fold Cross Validation

A special case of k-fold cross validation is 10-fold fold cross validation which splits the data set into ten subsets, allotting nine subsets to training and one subset to validation. As explained in the previous section a classifier is trained using 90% of the data leaving 10% for validation. The data is then shifted so the last 10% of the training data is validation data and the previous validation data is the first 10% of the training data. The classifier is then retrained on the newly distributed data set. This is repeated until all data has been used as training data. The performance of the model can be evaluated to see that the model is an appropriate fit to the data and the best model can be chosen for the classification task. If the model is well fit to the data each performance measure from each training fold should be reasonably close, this shows that the model always converges to a similar solution during training. Since ten classifiers have been
trained and evaluated during 10-fold cross validation, ten classifiers are available to choose between for best performance, even though there performance should be very close, one will most likely outperform the others and should be chosen for implementation.

1.5.3.3.4 Introducing a Test Set into 10-Fold Cross Validation

Since validation data is used during training to avoid over fitting by monitoring its performance in relation to the training data it affects the performance measure of the network through the error rate appearing smaller than the true error. By holding out a third data set as testing data set a model can better evaluated (Gutierrez-Osuna), (Parker, 2006). The advantage of the testing subset of data is that the data is not used to select a model within a training period. In other words the testing set is not used to evaluate the performance of the network until a final model is chosen within a train cycle (one of the ten folds). When using 10-fold cross validation along with a test set the data is still sectioned into ten subsets as seen in the previous section with 80% of the data being used for training, 10% for validation, and 10% for testing.
1.5.3.4 Measuring Performance

To measure performance during the neural network-training phase the mean-square error is used as the primary performance measure. Mean–square error (MSE) measures the average of the squares of the errors found in the model’s output with respect to the expected outcome. The error is the amount the expected output of the model differs from the actual outputs of the model. To generate an error value with the same units as the outputs the square root of the MSE can be taken to give the root-mean-square error (RMSE). Smaller values of MSE or RMSE indicate a better fit of the model to the data. The expression for MSE is shown below where $\hat{\theta}$ is the target output of the model and $\theta$ is the actual output.

$$MSE(\hat{\theta}) = E\left[ (\hat{\theta} - \theta)^2 \right]$$

(8)

1.5.3.5 Back Propagation Learning

As stated before, neural networks must in some way be trained in order to embed knowledge into their weights. The most common algorithm for training a neural network is through back propagation. The back propagation algorithm uses gradient decent to update weights in order to minimize the mean squared error (or other performance measurement) between the output of the network and the expected values of target outputs of the network (Parker, 2006). To update the weights, the contribution of each weight to the overall error is found by taking the partial derivative of the error function with respect to the weights. Through gradient decent, each weight is adjusted until the error has met a prescribed threshold. Another stopping mechanism, besides reaching a minimum error, is validation stopping as explained in section 1.5.3.2 For a more information on the specifics of the back propagation algorithm refer to (Leverington, 2009) and (Heaton, 2008).
1.6 Previous Works

Pitch detection is a rich and vibrant field in the sound-processing field. There have been many techniques that have been presented since the 1970’s, yet pitch detection; particularly multiple fundamental frequency pitch detection remains an unsolved problem. Three algorithms were chosen to study and evaluate the purposed algorithm against; these algorithms are all considered be state of art pitch detectors and are widely used for performance evaluation of pitch detection algorithms.

1.6.1 YIN Algorithm

The YIN algorithm is a fundamental frequency estimator that is designed to operate on monophonic speech and music signals. It is considered a highly accurate method and is a good baseline to compare against for monophonic signals. At its foundation it is a modified autocorrelation method that is optimized in order to prevent common errors (Chevereigne & Kawahara, 2001). Instead of using the autocorrelation function, which maximizes the product between an input and its delayed duplicate, the function used in the YIN algorithm minimizes the difference between an input and its delayed duplicate. After squaring the difference and averaging over a window, a value that minimizes the difference between the input and the delayed signal is searched for. This value corresponds to the fundamental frequency. A cumulative mean normalized difference function is used in order to reduce errors from imperfect periodicity. The next steps in the YIN algorithm involves reducing errors further through setting an absolute threshold on the calculated difference, as well as through parabolic interpolation to compensate for a period of the input not being a multiple of the sampling frequency. Finally a best local estimate is used in order to insure estimates remain stable over the duration of a single fundamental frequency.
As mentioned above, the YIN algorithm is designed to work for monophonic signals only, but is useful as a baseline comparison to compare the purposed multiple fundamental frequency detector’s performance for monophonic signals.

1.6.2 Multiple Fundamental Frequency Estimation Based on Harmonicity and Spectral Smoothness

Klapuri devised an algorithm in (Klapuri A., 2003) to separate multiple fundamental frequencies (F0) based on an iterative approach to subtract out an estimated F0 from a signal and repeat the process to detect multiple F0s. One notable attribute of this algorithm is its ability to process a wide range of signals through the use of spectral whitening, which suppresses timbral information associated with physical attributes of an instrument of acoustic environment that can affect a musical signal’s harmonic content. Klapuri’s system is a frequency domain technique which uses a discrete Fourier transform of a hamming windowed framed to transform time-domain acoustic signals into frequency domain signals used by the multiple F0 estimator. Its basic operation is shown in Figure 1.8, with the heart of the system being an iterative process that removes estimated F0s and related partials from a signal through an iterative process then estimates a new F0 with each iteration. Iteration stopping is provided through an estimation of the signal’s polyphony based on the previous F0 estimate and harmonic content of the signal. Klapuri’s algorithm has been shown to provide high accuracy in estimating both multiple fundamental frequencies, and estimating polyphony (Klapuri A., 2003). Although the algorithm provides highly accurate results, it is computationally expensive due to the large frame size needed for low frequency resolution when obtaining the frequency spectrum(Klapuri A., 2003).
1.6.3 Computationally Efficient Multi-pitch Analysis Based on Auditory Model

Tolonen and Karjalainen developed a computationally efficient model (known as “TK”) for multiple fundamental frequency detection based on a simplified auditory model that divided the input signal into two channels, then using an autocorrelation function, extracted relevant pitch information (Tolonen & Karjalainen, 2000). The model’s block diagram can be seen in Figure 1.9.

Figure 1.8: Block diagram of Klapuri’s multiple fundamental frequency estimation system
The first stage of the analysis involves a pre-whitening stage in order to separate the effect of a signal’s acoustic and physical environment on its frequency response. This is performed using a warped linear prediction (WLP) filter; more on this can be found in (Laine, Karjalainen, & Altosaar, 1994). The pre-whitened signal is then split into two separate channels, with the highpass channel being half-wave rectified which corresponds to the detection of the envelope of the signal. The periodicity detection stage is based on a generalized autocorrelation function that whose equation can be seen in eq.1 where $x$ is an input from a channel, $\alpha$ is a parameter that control frequency domain compression, and $r$ is the output from the generalized autocorrelation function.

$$r = \text{IDFT} \left( \text{IDFT}(x)^\alpha \right)$$  \hspace{1cm} (1)$$

Once the generalized autocorrelation function for each channel is computed they are summed together to form the so-called summary autocorrelation function (SACF). From there, the SACF is further processed to reduce redundancies by clipping the SACF to only positive values and scaling it in time by a factor of two, and then subtracting the clipped and scaled SACF from the
original SACF. This serves to remove repetitive peak, and can be repeated to remove higher order repetitive peaks from the SACF.

While the TK model has been shown to be highly computationally efficient (Tolonen & Karjalainen, 2000), it has also been shown to be inferior in recognizing monophonic pitches, and signals with a high level of polyphony (Klapuri A., 2003).
CHAPTER 2: METHODOLOGY

2.1 Multiple Frequency Pitch Detection using Multi-Class Neural Network Classifiers

This chapter presents a new algorithm for extracting pitch information of guitar signal that feature multiple fundamental frequencies. The algorithm presented makes use of binary-trees whose leaves contain neural networks classifiers in order to make classifications across many classes. An error correction scheme is also presented that is derived from communication theory error correcting codes. The algorithm can be broken into three major stages including preprocessing, multi-pitch classification, and error correction seen in Figure 2.1.

![Figure 2.1: Multiple fundamental frequency algorithm overview](image)

2.2 Preprocessing

2.2.1 Fast Fourier Transform

All processing involved with this algorithm is performed in the frequency domain. Since the goal is to identify multiple fundamental frequencies, it was intuitive to perform processing in the frequency domain due to the fact that in the frequency domain individual frequencies present in the signal will appear as clearly defined spikes. To perform the time to frequency domain transform, a fast Fourier transform (FFT) is used. A window size of 1024 samples for the transform was chosen; at a sampling rate 44.1 kHz, this corresponds to 23.2 milliseconds. A hop size of 256 samples was used to decrease the overhead of reading in the 1024 samples.
The FFT is a fast algorithm for computing the discrete Fourier transform (DFT). The DFT is defined in equation 2.1, where $X_k$ is the discrete frequency representation of $x_n$ over the time interval $n=0$ to $n=N-1$ (Kamen & Heck, 2007).

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi k n}{N}}, k = 0,1,\ldots,N-1 \tag{2.1}$$

Computing the DFT directly requires $O(N^2)$ operations while the FFT algorithm used here achieves $O(N\log N)$ operations using a radix-2 algorithm, where ‘O’ is big O notation that denotes the upper bound of an algorithms complexity. For more information, the reader should refer to (Phillips & Parr, 1999).
2.2.2 Important FFT Data

All the important frequency information in the musical signal is bound to the lower frequency region of the frequency spectrum with the majority of the harmonic content located between approximately 80 Hz and 5 kHz. This range corresponds to approximately the first 128 frequency bins of the FFT output. Before further processing is performed, the FFT output is reduced to only the first 128 bins; this not only highlights important data, but also cuts down the FFT signal size from 1024 bins to 128 bins. The range of harmonic content is illustrated in Figure 2.2 and Figure 2.3 showing both the upper and lower bounds of notes investigated in this paper.

Figure 2.2: Frequency Spectrum of total frequency range of FFT output (blue) and reduced frequency output (red) for upper bound of target F0 frequencies (~517 Hz)
2.2.3 Normalizing Data for Neural Network Processing

After the frequency transform is computed and relevant samples are extracted from the FFT output, the FFT is normalized before being processed by the neural network. The output from the FFT is normalized to a range of -1 to 1 to coincide with the range of the hyperbolic tangent activation function used in the neural network processing stages as shown in 1.5.1.5. The inputs are scaled using Equation 2.2 to map corresponding inputs between values of -1 and 1, where $x_{\text{norm}}$ is the normalized vector of the input $x$, $\text{min}$ is the minimum operator, and $\text{max}$ is the maximum operator. The result of normalization can be seen in Figure 2.4.

Figure 2.3: Frequency Spectrum of total frequency range of FFT output (blue) and reduced frequency output (red) for lower bound of target F0 frequencies (~86 Hz)
Figure 2.4: Magnitude spectrum after normalizing between -1 and 1

\[ x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)} \]  

(2.2)

2.3 Multiple Fundamental Frequency Classification

To identify a wide range of both individual and multiple combinations of notes, a multi-class classification system was developed based on three separate classifiers, each tasked with making multiple classifications through a neural network binary tree type architecture. The first classification determined the first note played in a sequence of combined notes (chord). This classification revealed which of the twelve possible musical notes was played as well as what octave it occurred in (in terms of this research a range of three octaves were considered). The second classification gave an estimate of how many notes occurred (one to six). By combining these two classifications, the total number of possible chords that the third classifier must classify is greatly reduced; in terms of this research, root note classification reduces complexity
by a factor of 36, while estimating the number of notes further reduces the complexity by a factor of 6. After the chord recognition has been performed, an error-correcting scheme is applied to the output. This scheme is similar to communication systems error correction coding (Proakis & Masoud, 2002). An over-view of the system is presented in Figure 2.5. It should be noted that the output of the algorithm produces note values corresponding to standard MIDI frequency values that are quantized between 0 and 127 as reviewed in 1.4.

![Figure 2.5: Block Diagram of multiple-F0 classification algorithm](image)

**2.3.1 Classification of Single Notes and Root Notes**

The first classification made in the process of identifying music notes and chords was that of classifying the root note of the chord or lowest frequency note to appear in a series of combined
notes. It should be noted that in terms of music theory, the root note is not always the lowest note in terms of frequency to appear in a series of notes played simultaneously, but is often the note that other notes are built around based on the musical key that is being played (Hal Leonard Corporation, 2004). For this paper, the root note will be considered the lowest note in terms of frequency to appear in a chord. A series of artificial neural networks (ANN) arranged in a binary tree are used for classification. More information on using neural network binary trees for multiple class classification problems can be found in 1.5.2.1. In terms of network structure, there is a single hidden layer of each network that has a weight matrix consisting of 1-by-128 weights that are activated by a saturated linear function that approximates the hyperbolic tangent function.

2.3.1.1 Binary Tree Classification for Root Note Detection

The classification is built around several neural networks arranged in a binary tree type of structure as discussed in 1.5.2.1. This type of arrangement performs multiple binary classifications in order to achieve a highly accurate overall classification. Each binary classification is based on choosing a frequency range the input belongs too, starting with a wide frequency range and iteratively narrowing the range down until there are only two possible frequencies left to choose between. This process is illustrated in Figure 2.6.
2.3.1.2 Network Architecture for Root Note Detection

Within each binary leaf of the binary tree is a neural network classifier that makes the decision on the path to take through the tree. Each neural network has a feed-forward architecture that has been trained through back-propagation as explained in 1.5. A block level view of the architecture is seen in Figure 2.7 and the transfer function in equation 2.3 where \( a \) is the output of the ANN, \( \mathbf{W}_{1,1} \) and \( \mathbf{W}_{2,1} \) are weight matrices for the input layer and output layer respectfully, \( \mathbf{b}_1 \) and \( \mathbf{b}_2 \) are bias terms, \( \mathbf{f}_1 \) is a modified saturated linear activation function and \( \mathbf{f}_2 \) is a linear activation function.
2.3.2 Polyphony Estimation

The second classification is made to estimate the total number of notes (amount of polyphony) played within a window. For this research a minimum of 1 note (monophonic) to a maximum of 6 notes (polyphony of six) is the range of the classification. A number of features were investigated including number of harmonic peaks and spectral power both, which provided a weak correlation to number of notes played, as it is evident in figures 1 and 2. Considering only the frequency spectrum, the network architecture was chosen with two hidden layers and an appropriate number of neurons to perform the polyphony estimation. A network design was implemented by first picking a small number of neurons in the hidden layers, then increasing the number of neurons until an acceptable error was achieved while maintaining an appropriate network size.

\[ a = f_2 \left( LW_{2,1} f_1 \left( IW_{1,1} \cdot x + b_1 \right) + b_2 \right) \]

\[ f_1 = \text{saturated-linear}(x) = \begin{cases} 
-1, & x \leq -1 \\
-x, & -1 < x < 1 \\
1, & x \geq 1 
\end{cases} \quad (2.3) \]

\[ f_2 = \text{linear}(x) = x \]
2.3.2.1 Network Architecture for Polyphony Estimation

The same basic feed-forward ANN trained with back-propagation learning is used for polyphony estimation as that used for the root note selection networks. However, there is a much more complex relationship between the numbers of notes played compared to extracting a fundamental frequency, so a more complex network must be used to achieve good performance. It was found that a network with two hidden layers, with a size of ten and one respectively achieved good performance. A block level view of the architecture is seen in Figure 2.8 and the corresponding transfer function in equation 2.4.

![Block diagram view of each feed-forward network in binary decision tree from root note recognition](image)

Figure 2.8: Block diagram view of each feed-forward network in binary decision tree from root note recognition

\[
\begin{align*}
a &= f_3 \left[(LW_{3,2}f_2[LW_{2,1}f_1(IW_{1,1} \cdot p + b_1) + b_2] + b_3)\right] \\
f_1, f_2 &= \text{saturated-linear}(x) = \\
&\begin{cases} 
-1, x \leq -1 \\
-x, -1 < x < 1 \\
1, x \geq 1 
\end{cases} \\
f_2 &= \text{linear}(x) = x
\end{align*}
\]  

(2.4)

2.3.2.2 Skewed Binary Tree Classification for Polyphony Estimation

A slightly modified version of the ANN binary tree was used to classify the amount of polyphony. Since the total amount of polyphony is not a power of two, an evenly distributed
binary tree cannot be used and must be skewed. This can be seen in Figure 2.9 where the input is a two-note chord, so the polyphony chosen in the binary classification tree is two.

![Figure 2.9: Skewed binary classification tree showing an example of data flow when a two-note chord is presented at the input.](image)

2.3.3 Chord Recognition

The chord recognition uses a similar process to recognize musical chords or groupings of multiple fundamental frequencies as estimating root notes and amount of polyphony in terms of ANN architecture and the use of binary classification but groups different classes based on features identified through music theory such as major, minor, dominant, suspended etc. For this research, 80 chord combinations were chosen that represent a well-defined population of typical chords played on a guitar.

2.3.3.1 Network Architecture

Once again, a feed-forward ANN with back propagation learning is used as the base of the classifier system. All but the first classification layers of the ANNs have the same architecture in terms of number of layers and weight matrix sizes as seen in Figure 2.7 for root note recognition with the corresponding transfer function seen in equation 2.3. The first classification proved to
be a much more complex problem when trying to separate chord types into two groupings so a network of the same number of layers and weight matrix sizes used in Figure 2.8 was used with the corresponding transfer function seen in equation 2.4.

2.3.3.2 Classification of Chords Based on Music Theory Principles

Once again a binary decision tree is used to perform the classification, this time it is used to classify which chord or grouping of notes has been played. This binary tree has been constructed through music theory principles on how chords are constructed as seen in Figure 2.10. For example, all major chords contain a major third interval, where all minor chords contain a minor third interval; a more in-depth discussion on how chords are constructed and arranged is discussed in 1.2.2. A separate tree is constructed for each polyphony group as well as each root note group. Though this increases design time, it significantly limits the number of chords needing classification, therefore increasing accuracy while decreasing the size of each ANN.
2.3.4 Post-Processing Error Correction

Further accuracy can be achieved during the post-processing phase after the ANN processing stage has processed the input data. A modified algorithm based on communication theory error detection and correction is used to increase the accuracy of the multiple-F0 detection algorithm presented in section 2.3.3. This error detection and correction algorithm is based on redundancy coding, an error correction code based on reviewing multiple outputs to decide if an error has been made and how to correct it. Error correction is achieved through a majority vote based on what output occurs most frequently.
2.3.4.1 Communication Theory Motivated Error Correction

For error detection and correction purposes, it can be useful to think of a classification system such as an ANN as a communications system. A basic communication system can be modeled as three separate subsystems as seen in Figure 2.11. These consist of a Transmitter that encodes and sends out a message, a channel to carry the message (radio waves, transmission line, fiber cable, etc.), and a receiver to decode and present the message to an end user. There is also channel noise caused from environmental factors that can corrupt the message received by the receiver. In a classification system the inputs and preprocessing stage can be thought of as a transmitter, the classifier can be thought of as a channel for communication, and the outputs can be treated as a receiver. Any errors produced by the neural network can be treated as channel noise that can be corrected by modifying techniques used in communications systems.

![Block diagram of basic communication system](image)

Figure 2.11: Block diagram of basic communication system

2.3.4.2 Repetition Coding

In communication systems, repetition coding is a form of error correction coding that sends the same message multiple times to ensure the correct message is received (Proakis & Masoud,
2002). For example if the message is ‘1’ and the repetition rate is three, then the transmission sent through the channel would be ‘111’. If the message received is ‘011’ then one error has been detected and can be corrected through majority vote, correcting the corrupted message to ‘1’. If more bits are corrupted, for example if the message in the previous example is transmitted as ‘010’, then an error is detected but the wrong message is extracted from the transmission as ‘0’. To add accuracy to the redundancy detection, more bits can be added to the transmission; for example, five bits would be able to detect and correct a two-bit error.

A modified version of the repetition-coding algorithm can be used to correct error in a classifier’s output. Instead of computing the same sample block by the ANN based classifier, the current sample block is compared to adjacent sample blocks by holding out the current sample block to see if an error has occurred. In order to maximize throughput, this error correction algorithm is only used when a change in output has been detected. Since musical notes have a predictable signal envelope, abrupt changes to a new note are not likely to happen. When a change in output has occurred, the next \( n \) outputs are compared to each other with the correct output determined by majority vote.
CHAPTER 3: RESULTS

The results presented in this section are the product of several experiments to gauge the performance of the multiple fundamental frequency pitch detector presented here. Results for each individual classifier as well as the entire algorithm are presented. These results are also compared to other algorithms, the first uses multiple IIR comb filters in order to estimate multiple pitches, while the other is an iterative approach based on spectral subtraction.

3.1 Measuring Performance

3.1.1 Accuracy and Precision

Accuracy can be defined as the overall correctness of the model and can be expressed as the total number of correct classifications divided by the total number of classifications. Precision is a measure of accuracy, given a specific class has been predicted. Precision is calculated using the number of true positives divided by the sum of true positives and false positives as seen in equation 1. A true positive is a number defining the classification that a predicted class has been correctly identified, while as a false positive is a number defining that a predicted class has been incorrectly identified. A true negative refers to a number defining a class that is not being classified is correctly rejected, while a false negative corresponds to the number other classes that were incorrectly identified as the class under test. The error rate is defined as the overall number of instances that were incorrectly classified, and is 3.1 subtracted by the accuracy as defined in equation 3.2.
3.1.2 Recall (Sensitivity) and Specificity

Recall or sensitivity is a measure that expresses the proportion of correctly classified classes that are properly identified, while the measure of specificity expresses the proportion of other classes that are correctly identified. Their mathematical representations can be seen in equations 3.3 and 3.4.

\[
\text{recall} = \frac{\text{true positives}}{\text{true positives + false negatives}} \tag{3.3}
\]

\[
\text{specificity} = \frac{\text{true negatives}}{\text{true negatives + false positives}} \tag{3.4}
\]

3.1.3 Confusion Matrix

A confusion matrix is a table or plot that allows for a visual assessment of a classification algorithm with columns representing the actual class and rows representing a predicted class. A good classifier will have the highest values falling along the diagonal of the table or plot. When viewed as a plot or image, the diagonal from top left to bottom right should contain the most “intense” colored values that represent high numbers. Figure 3.1 shows an example of a confusion matrix in table form with its corresponding graphical representation. True positives,
true negatives, false positives, and false negatives can directly be extracted from the confusion matrix; given a class \(c_{m,n}\), where \(m\) is the row location and \(n\) is the column location, the true positive value of class \(c\) is defined as \(c_{c,c}\), true negatives of a class is defined as the sum of all values along the diagonal except the class under test, false positives of a class are defined as the sum of row values \(c_{c+1,c}\) to \(c_{c+C,c}\), where \(C\) is the size of the confusion matrix, false negatives of a class are defined as the sum of row values \(c_{c,c+1}\) to \(c_{c,c+C}\). These values also link the confusion matrix directly to the performance measures discussed in sections 3.1.1 and 3.1.2.

![Confusion Matrix Table and Plot](image)

Figure 3.1: Comparison of example confusion matrix table and plot

### 3.2 Results of Root Note Detection

Below are the results for the root note detection phase. Presented first in Figure 3.2 is the confusion matrix for the root selection classifier. It should be noted that the values have been logarithmically scaled in order to show where misclassifications have occurred. Following the confusion matrix results are histograms showing the precision, recall (sensitivity), specificity, and miss rate for root note detection in Figure 3.3, Figure 3.4, Figure 3.5, and Figure 3.6 respectfully. To summarize the results, the average, minimum, and maximum values for each performance measurement is summarized in Table 3.1.
Figure 3.2: Confusion matrix for root selection classifier
Figure 3.3: Histogram of precision values for root selection classifier

Figure 3.4: Histogram of recall (sensitivity) values for root selection classifier
Figure 3.5: Histogram of specificity values for root selection classifier

Figure 3.6: Histogram of error rates for root selection classifier
Table 3.1: Summary of performance statistics for root note selection classifier

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.993944</td>
<td>0.995828</td>
<td>0.999925</td>
<td>0.00605625</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.9862</td>
<td>0.9848</td>
<td>0.9996</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.0138</td>
</tr>
</tbody>
</table>

3.3 Polyphony Estimation Results

Below are the results for the polyphony estimation phase. Presented first in Figure 3.7 is the confusion matrix for the polyphony estimation. It should be noted that the values have been logarithmically scaled in order to show where misclassifications have occurred. Following the confusion matrix results are bar plots showing the precision, recall (sensitivity), specificity, and miss rate for each of the 6 polyphony classes in Figure 3.8, Figure 3.9, Figure 3.10, and Figure 3.11 respectively. To summarize the results, the average, minimum, and maximum values for each performance measurement is summarized in Table 3.2.
Figure 3.7: Confusion matrix for polyphony estimation
Figure 3.8: Plot of precision values for polyphony estimation

Figure 3.9: Plot of recall (sensitivity) values for polyphony estimation

Figure 3.10: Plot of specificity values for polyphony estimation
3.4 Chord Recognition Results

Below are the results for the chord recognition phase. Four separate results are presented for chord recognition; first is the case where it is assumed polyphony is known a priori, the second when the input is processed by the root note detection classifier and the polyphony estimation classifier, and finally results for an error corrected output are presented with a parity check of three and five. Presented first in Figure 3.2 is the confusion matrix for the polyphony estimation.
It should be noted that the values have been logarithmically scaled in order to show where misclassifications have occurred compared to knowing polyphony a priori and when polyphony is estimated. Following the confusion matrix results are histograms showing the precision, recall (sensitivity), specificity, and miss rate for each of the 80 chord classes in Figure 3.13, Figure 3.14, Figure 3.15, and Figure 3.16 respectfully. To summarize the results, the average, minimum, and maximum values for each performance measurement is summarized in Table 3.3.

Figure 3.12: Confusion matrices for chord recognition when polyphony is known a priori (left) and is estimated (right)
Figure 3.13: Histogram of precision values for chord recognition; Polyphony known a priori (red/grey), polyphony estimated (blue)

Figure 3.14: Histogram of recall (sensitivity) values for chord recognition; Polyphony known a priori (red), polyphony estimated (blue)
Figure 3.15: Histogram of specificity values for chord recognition; Polyphony known a priori (red), polyphony estimated (blue)

Figure 3.16: Histogram of error rate values for chord recognition; Polyphony known a priori (red), polyphony estimated (blue)
Table 3.3: Summary of performance statistics for chord recognition for polyphony known a priori and estimated polyphony

<table>
<thead>
<tr>
<th>Polyphony</th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
<th>Miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>known a priori</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.995369</td>
<td>0.997464</td>
<td>0.9999725</td>
<td>0.00463125</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.9662</td>
<td>0.9682</td>
<td>0.9997</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.0338</td>
</tr>
<tr>
<td>Estimated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.979751</td>
<td>0.989553</td>
<td>0.999865</td>
<td>0.02047875</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.9508</td>
<td>0.9639</td>
<td>0.9996</td>
<td>0.0015</td>
</tr>
<tr>
<td>Maximum</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.0492</td>
</tr>
</tbody>
</table>

Presented next are the results showing the improvement through the use of error correction.

Figure 3.17 shows the confusion matrices for the error corrected outputs when both three and five frame comparisons are used for error correction. Table 3.4 illustrates the overall performance statistics for error correction network.
Figure 3.17: Confusion matrices for error corrected output when three comparisons are used (left), and five comparisons are used (right).

Table 3.4: Summary of performance statistics for chord recognition for polyphony known a priori and estimated polyphony

<table>
<thead>
<tr>
<th>Comparisons</th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
<th>Miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Average</td>
<td>0.990908</td>
<td>0.99434</td>
<td>0.99996375</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>0.9723</td>
<td>0.9685</td>
<td>0.9998</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Average</td>
<td>0.994106</td>
<td>0.998083</td>
<td>0.99989125</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>0.9785</td>
<td>0.9833</td>
<td>0.9998</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3.5 Comparison to Other Techniques

The proposed multiple fundamental frequency detection algorithms were compared to the YIN algorithm to evaluate its performance for classifying single notes, and two multiple fundamental
frequency estimators; the first method is based on an auditory model while the second method is based on an iterative approach based on spectral subtraction. The results were compared to those in (Klapuri A., 2003), (Tolonen & M., 2000), and (Thomas, 2012). The results compare error rates of each algorithm.

Figure 3.18: Comparison of error rates from different algorithms for detecting fundamental frequencies

3.6 Benchmarks on Embedded Platform

To evaluate the potential for implementing the proposed algorithm in a real-time system to be used for music performance, each function of the algorithm was benchmarked to evaluate its execution time. The benchmarking was performed on an Analog Devices SHARC-21469 processor clocked at 450MHz. This processor is a digital signal processor (DSP) that reflects what would be used in a commercial system. Below in Table 3.5 benchmarks for preprocessing, root-note detection, polyphony estimation, chord recognition, as well as an estimate of the overall execution time are presented if the system was implemented in a serial fashion.

Table 3.5: Execution times each function of the proposed algorithm
<table>
<thead>
<tr>
<th>Process</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Read</td>
<td>5.60</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>1.04</td>
</tr>
<tr>
<td>Root Note Detection</td>
<td>0.03</td>
</tr>
<tr>
<td>Polyphony Estimation</td>
<td>0.25</td>
</tr>
<tr>
<td>Chord Recognition</td>
<td>0.10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7.02</strong></td>
</tr>
</tbody>
</table>
CHAPTER 4: DISCUSSION

4.1 Root-Note Detection

It can be seen that the note detection classifier performed with high accuracy as evidenced in section 3.2. Figure 3.2 clearly shows a high level of accuracy in the classifier with the majority of classification falling along the diagonal of the confusion matrix. It can also be shown that any misclassifications are sparsely distributed and are a large distance away from the intended classifications. This can be attributed to errors occurring at the beginning of the binary tree and shows that the classifier has trouble differentiating between two large classes that have a wide range of data. Though these errors can be considered serious, they are rare, as shown in the histograms for precision, recall, specificity, and error rate in Figure 3.3, Figure 3.4, Figure 3.5, and Figure 3.6 respectfully. The very high percentages for precision, recall, specificity, and very low miss rate as illustrated in the aforementioned figures and Table 3.1 are informative in showing this model is very suitable for identifying root notes from single notes and chords produced by guitars.

4.2 Polyphony Estimation

The polyphony estimation classifier also performed with high accuracy, which is evident in the results presented in section 3.3. The errors produced from the classifier were observed to increase with the amount of polyphony, as seen in the confusion matrix in Figure 3.7; this was expected since there are little measurable differences between the amounts of polyphony, particularly at higher levels. The five-string class had the worst performance as evident in Figure 3.8, Figure 3.9, Figure 3.10, and Figure 3.11, although, with a maximum error rate of approximately 3%, the error rate can be considered acceptable with the majority of misclassified
instances falling within the adjacent classes. Again, high levels of precision, recall, specificity, and a low error rate show the model is a good fit to the data.

4.3 Chord Recognition and Error Correction

The chord recognition classifier showed good performance, particularly when polyphony is known a priori. It can be seen in Figure 3.12 that the majority of misclassification occurs in classes that correspond to chords with complex harmonic structure, such as suspended, augmented, and diminished. It should also be noted, like in root detection, that these errors can occur at a large distance away from the expected value; this illustrates that errors take place within the first classification branch of the binary tree. When polyphony is estimated, their misclassifications follow a similar pattern when polyphony is known a priori but occur at a greater frequency as evident in the plot of the error rate in Figure 3.17. It can be seen in Figure 3.17 that a significant improvement in performance is achieved in chord recognition when applying error correction. By looking at the confusion matrices, it can be seen that errors were smoothed out by holding out samples for comparison when a change in output is detected. Table 3.4 and Table 3.5 show high values for precision, recall, specificity, and low values for miss rate, indicating the chord recognition modal is a good fit to the data, while it is also evident that there is an improvement seen when error correction is applied to the output.

4.4 Comparing Algorithms

When compared to other algorithms the algorithm, proposed gave overall better results in terms of error rate as seen in Figure 3.18. Klapuri’s harmonicity and spectral smoothness algorithm performs the best out of the algorithms and is close in performance when compared to the proposed algorithm up to a polyphony level of 3. This is most likely attributed to Klapuri’s
polyphony estimation stage’s performance at higher polyphony’s than to the algorithms fundamental frequency estimation stage.

4.5 Real-Time performance

Real-time performance was evaluated by individually benchmarking each function of the overall system in order to evaluate weaknesses and see where performance can be increased. It can be seen in Table 3.5 that the total execution time is well under the standard audio latency time of 12ms. Though these measurements are not completely precise in terms of timing, as the execution time functions inherently have some overhead, and in practical implementation the functions would most likely be integrated into a real-time operating system to maximize throughput, these results provide good insight into the ability for using neural networks classifiers for pitch detection in a real-time environment.
CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusions

A new algorithm has been presented for multiple fundamental frequency detection using multi-class neural network classifiers that shows strong performance in terms of accuracy of classification as well as speed performance. The algorithm made use of multiple classifiers to identify the root note of a chord and the amount of polyphony, which were used to select the third classifier to classify what groups of notes were played. The algorithm was shown to work on a wide variety of musically combined notes played by a guitar. When compared to other algorithms used to estimate multiple fundamental frequencies it was illustrated that the proposed algorithm performed better under similar test conditions. Finally the algorithm showed great potential for operating in a real-time system where musical performance is the application.

5.2 Future Work

It was shown that the proposed algorithm worked well under the given test conditions, but there is room to expand for a wider variety of operating conditions. The algorithm should be expanded to work for multiple input sources besides guitar, this would lend to its use for music transcription applications. To avoid the need for an increase in training examples, spectral whitening, as used in other methods mentioned in this paper should be investigated as a means of designing an algorithm that can easily be implemented for multiple instruments. The number of possible note combinations should also be expanded, while the proposed system has been trained to recognize the majority of musically relevant guitar notes and chords, a more flexible system may be realized by combining the root-note and polyphony estimation of the proposed algorithm with Klapuri’s spectral subtraction; considering the efficiency at which both the root-note detection and polyphony estimation classifiers operate an iterative subtraction should be feasible.
Finally in order to fully implement the system, integration into a real-time operating system or kernel should be considered in order to improve real-time performance in terms of timing and predictability.
BIBLIOGRAPHY


CURRICULUM VITAE

Nathan K. Hilbish was born May 12, 1987, in Lynchburg, Virginia, and is an American citizen. He graduated from Jefferson Forest High School, Forest, Virginia in 2006. He received his Bachelor of Science in Electrical Engineering at Virginia Commonwealth University in 2010.