Linearity One Final Project Paper

Alison Palmer and Anusha Datar

May 9, 2018

1 Introduction and Background

For our final Linearity One project, we set out to create a program that recognizes the notes and octaves associated with musical notes and chords when given an audio sample that contains only that note or chord. The purpose of this project was to learn about the theory of the Discrete Fourier Transform and the Fast Fourier Transform and the application of these procedures to signal processing. To start the process, we first created a program to recognize a single note. We then moved onto recognizing chords.

1.1 Introduction to Music Fundamentals

Music can be broken into seven elements: rhythm, dynamics, melody, harmony, tone color, texture, and form. For our application, we will primarily focus on melody and harmony. Melody can be broken down into the presence and order of pitch in music. Pitch is the sensation of frequency of sound. While frequency and pitch are correlated, i.e. while high pitch implies that the signal has a high frequency, they have no direct relationship. Pitch can also be defined as the level of tone. Tone is a periodic sound that repeats. Frequency is how often a wavelength repeats given a set period of time. Harmony is the collection of various pitches played simultaneously to create a chord. A chord progression is a pattern of chords played after one another. Generally, the western musical scale assigns the notes A through G to steps based on predetermined frequency intervals on a sliding scale. Each iteration of this scale is an octave. Generally, the central octave is the fourth octave, where C_4 corresponds to Middle C.

1.1.1 Notes and the Frequency Domain

As long as temperament is equal and standard (i.e. the tuning matches usual intervals), the frequency of a given sound directly maps to the note and octave value. Generally, the value of the note's frequency is tuned such that A_4 is set to a particular frequency and other notes are staggered at steps above and below this value. A very common scale places A_4 at 440 Hz and staggers the notes above and below accordingly. This allows for straightforward and robust note classification based exclusively on noise filtering and frequency.

1.1.2 Chords and the Frequency Domain

A chord is made up of three or more discrete notes played simultaneously. Chords are made up some principle root note. Each major. Each chord has an associated note and specification associated with it. For example, C major is made up of the root note C, the third (E), and the fifth (G). While the sound that combining these three notes blends cleanly in the time domain, the separate frequencies remain fairly detectable because of how the individual notes are played. At the same time, harmonics associated with physically playing the note and blending that occurs distorts the exact peaks.

1.2 Introduction to Fourier Transform

1.2.1 Fourier Series

Any function can be represented as a series of sines and cosines. The Fourier Series can be describes through either trigonometric terms or exponential terms. Both definitions are listed below. See the appendix for the

equality proof.

$$f(x) = a_0 + \sum_{n=1}^{\infty}(a_n * cos(n\omega_0 t) + b_n * sin(n\omega_0 t)) \tag{1}$$

$$f(x) = \sum_{n=1}^{\infty} c_n * e^{in\omega_0 t} \tag{2}$$

The basis for the Fourier Transform is rooted in the idea that any function can be expressed given sines and cosines. Given a Fourier series, the Fourier Transform can extract the amplitude of the various waves in order to use them for analysis. In our case, we are extracting those amplitudes to first identify notes, and then chord patterns.

1.2.2 Fourier Transform

The basic purpose of the Fourier Transform is to convert a function from the time domain to the frequency domain. This means that the signal or function is broken down into the frequencies that it is made of. The Fourier transform is mathematically defined below:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \tag{3}$$

From this formula, we can extract the real and imaginary components which are defined below.

$$F(\omega) = \int_{-\infty}^{\infty} f(t)cos(\omega t)dt + i\int_{-\infty}^{\infty} f(t)sin(\omega t)dt \tag{4}$$

The magnitude of F is known as the amplitude spectrum of f(t). The phase of F gives you the phase spectrum of f(t).

1.2.3 Discrete Fourier Transform

The Discrete Fourier Transform is the evaluation of a finite data set using the Fourier Transform method. The Fourier Transform evaluates a continuous and infinite data set. In practical application of the Fourier Transform, the Discrete Fourier Transform is used. Its mathematical definition is below.

$$F(k) = \sum_{n=0}^{N-1} f(n)e^{-i\omega_k n} \tag{5}$$

In order to evaluate the signal, a method called windowing is used. Instead of evaluating the entire sample, only a portion of the signal is used to perform the transformation.

1.2.4 Fast Fourier Transform

While the Discrete Fourier Transform is certainly a useful framework, the algorithmic complexity required to calculate the value is higher than desirable. Using the DFT, the complexity is $O(N^2)$, where N is the number of points sampled.

The Fast Fourier Transform (FFT) is a more efficient and often more accurate algorithm to compute a Discrete Fourier Transform over the course of a single sample. It is used in programming computation in order to save time and resources. There are many implementations of this algorithm, but the most commonly used one, and the one leveraged in this project, is called the Cooley-Turkey algorithm. It is a recursive divide-and-conquer implementation algorithm that removes much of the redundancy associated with a direct DFT. For mathematical background and pseudo code, see the appendix.

The FFT enables significantly faster computation as compared to the normal Discrete Fourier Transform. By splitting the sample, the time complexity of this algorithm is only $(Nlog_2N)$. For example, with a 2048 point FFT, this decreases the overall number of individual computations from 4194304 to 22528! However, there are a few constraints necessary for this algorithm to be functional: the number of samples must be a

power of two and each sample should ideally be a single sample of a periodic waveform. In regards to the first constraint, the user specifies the number of samples, so this problem generally has no impact if included in the measurement. If this is not the case, then the procedure will still provide some value, but it will also contain spectral noise. Strategies to mitigate this noise include windowing (oversampling from the center of the sample and removing discontinuities from the ends) and overlaying multiple FFTs and finding the smoothed value.

2 Implementation

To implement chord and note recognition using the Fourier transform, we chose to use the Python programming language and several functions associated with the libraries numpy and wave. The reason we chose to use this particular set of tools was that Python allowed us to quickly write scripts to test frameworks and methodologies. In addition, numpy and wave provide clean and convenient implementations of various signal processing techniques such that we could easily construct our implementation without sacrificing the ability to customize our approach.

2.1 Note Recognition

Our general approach to note recognition was to capture the incoming signal, utilize signal processing techniques to determine the dominant frequency of the input, and then reference a data structure with known frequencies and notes based on a specified tuning scheme to determine the value and octave of the note itself. We defaulted to a modality where A4 was 440Hz.

2.1.1 Frequency Extraction Technique

To find the frequency value of the given sample, we sample chunks of the signal and use a Blackman window to find the largest frequency value. Then, we employ quadratic interpolation to determine the value of the largest frequency. After averaging several values, the program chooses the closest frequency in the given table of known notes and octaves and returns that value.

2.1.2 Results and Accuracy

To provide some metric regarding our performance, we included a confidence calculation within the note recognition algorithm: for each calculated note, the confidence calculation assumes that the guess is correct and determines the error between the calculated frequency and the true frequency of the actual note. Three octaves above and below C major, we had 100% accuracy and 99% confidence.

2.2 Chord Recognition

To recognize chords, the input file gets put through the FFT. From this, a plot is generated with the x axis representing frequency and the y axis represented the magnitude of it's presence in the chord. From this graph, we utilize mode and quadratic interpolation to pull the top three peaks. These three peaks represent the three notes that make up the chord. During this process, we also filtered out any frequency above 20,000Hz due to the fact that humans cannot hear frequencies that high.

When trying to get our chord recognition to work, we ran into some problems. Our data set consisted of chords played on an electric guitar. Due to the shape of a guitar and the way that is is played, the chord FFT does not cleanly include peaks for the three most common frequencies. For that reason, we needed to use existing datasets to model how the noise and blending behaved in the frequency domain. As a result, we chose to the K-Nearest Neighbors machine learning algorithm in order to accurately identify the notes that make up the chord being sampled.

2.2.1 K-Nearest Neighbors Machine Learning Algorithm Method

The K-Nearest Neighbors Algorithm uses approximate distances between data points and their classification to classify future data points. For the training material, we used electrical guitar chords. When a new data

point is being evaluated, the algorithm finds the closest point and assigns the class from that value to the new data point. We chose to compute the distances between our test point and each point based on pure Euclidean distance between the most common frequencies in the given file; the reason we made this choice is because it provides the most clean mapping while sacrificing the least data. At the same time, this method is limiting because it falsely rewards points being in the correct order. While specific frequencies should appear in order, the fact that some values are very close together yields potential error. In the future to make our program more accurate and advanced, we would use some type of weighting system to better place the new data points along the given data points.

2.2.2 Results and Accuracy

Our implementation of K-Nearest Neighbors proved to have 100% accuracy on a given test set of major chords with the average distance from the training value close to zero. While this is certainly a promising result, using a more diverse and large test set, and likely supplementing this with a more extensive training set, is necessary to validate this method.

A Appendix

A.1 Equation 1 = Equation 2

Given:

$$f(x) = a_0 + \sum_{n=1}^{\infty}(a_n * cos(n\omega_0 t) + b_n * sin(n\omega_0 t))$$

and

$$f(x) = \sum_{n=1}^{\infty} c_n * e^{in\omega_0 t}$$

prove that they are equal:
Euler's Formula is defined as

$$e^{i\theta} = cos\theta + isin\theta$$

therefore

$$\sum_{n=1}^{\infty} c_n * e^{in\omega_0 t} = \sum_{n=1}^{\infty}(c_n(cos(n\omega_0 t) + sin(n\omega_0 t)))$$

$$= \sum_{n=1}^{\infty}(c_n * cos(n\omega_0 t) + c_n * sin(n\omega_0 t))$$

Let $a_n = c_n$ and $b_n = c_n * i$. Therefore

$$a_0 + \sum_{n=1}^{\infty}(a_n * cos(n\omega_0 t) + b_n * sin(n\omega_0 t)) = \sum_{n=1}^{\infty} c_n * e^{in\omega_0 t}$$

A.2 Cooley-Turkey Fast Fourier Transform

Consider a general interleaved data set that contains a periodic chunk of the signal to perform analysis on with a length that is some power of two. Over the course of this sample, split the data set based on odd and even indices (i.e. place all points with odd indices in one sample and all points with even indices in another) recursively and perform the DFT in place. Then, manually recombine the existing pieces into a full sample.

For pseudo code purposes, the signal is an array S with a length that is some power of two. Then, the FFT procedure is a short divide-and-conquer algorithm.

```
def FFT(S):
    # Split into even and odd blocks.
    X_even = FFT([S[0], S[2], S[4], S[6], S[8]...)
```

```
    X_odd = FFT(S[1], S[3], S[5], S[7]...)
    # Perform the actual computation.
    factor = exp(-2j * pi * range(S) / S)
    # Merge solutions.
    return concatenate([X_even + factor[:S / 2] * X_odd,
                        X_even + factor[S: / 2:] * X_odd])
```

A.3 KNN Algorithm

Below is the pseudo code for finding the K-Nearest Neighbor path using Euclidean distance.

```
def KNN(x):
    #X = training data previously defined
    #Y = Labels for training data
    #x = unknown data
    #create list to store distances
    list=[]
    #goes through all the training data and finds the distance
    #between the training data and the new point
    for (i in X):
        #adds the absolute value of the distance to the array
        list.append(abs(freq(x)-freq(i)))
    #returns the class associated with the closest training point
    return Y[list.min(list)]
```

In order for application to be more accurate, we utilized a weighted system to measure the distance between points.

B Sources

1. "Frequencies for equal-tempered scale, A4 = 440 Hz." Michigan Technical University. https://pages.mtu.edu/ suits/notefreqs.html.

   This frequency table contains a set of notes, octaves, and corresponding frequencies necessary for responsive note recognition.

2. Muller, M., Ellis, D., Klapuri, A., Richard, G. "Signal processing for music analysis," IEEE Selected Topics in Signal Processing (2011): accessed May 1, 2018.

   This paper directly connects signal processing and music analysis.

3. Jacobsen, Daniel Christopher. A Listener's Introduction to Music. Wm. C. Brown Publishers, 1992. See esp chap. 1, "The Elements of Music."

   This chapter provides a broad overview of music theory terminology and concepts that allow for a consistent and useful lexicon.

4. Lenssen, Nathan and Deanna Needell. "On the Mathematics of Music: From Chords to Fourier Analysis." arXiv.org, (2013): access May 2, 2018, https://arxiv.org/abs/1306.2859.

   AN interesting and useful consideration of how chords map to the signal domain.

5. Plack, Christopher J. and Andrew J Oxenham. Pitch : neural coding and perception. Springer Handbook of Auditory Research (24). Springer, New York, pp. 1-6. See esp. chap. 1, " Overview : The present and future of pitch."

   This chapter provides background information regarding the notion of pitch and its relationship with signal processing.

6. Richardson, Mark A, "Fundamentals of the Discrete Fourier Transform," Sound and Vibration Machine, March Edition(1978): accessed April 30, 2018, https://issuu.com/vibranttechnology/docs/paper09-fundamentalsofthedft.

   This paper provides a clear description of the mathematical procedure and significance of the discrete Fourier transform that was key to building an initial understanding of the discrete Fourier transform.

7. Tay, Bunheang Hyun, Jung Keun Oh, Sejong. (2014). A Machine Learning Approach for Specification of Spinal Cord Injuries Using Fractional Anisotropy Values Obtained from Diffusion Tensor Images. Computational and mathematical methods in medicine. 2014. 276589. 10.1155/2014/276589.

   This paper provided an overview of the K-Nearest Neighbors Algorithm, wh