Harmonic Analysis Using Neural Networks

Vincent Tsui & W. James MacLean
Edward S. Rogers Sr. Department of Electrical & Computer Engineering,
University of Toronto, Toronto, Canada, M5S 3G4

Abstract

This paper describes *Harmonic Analysis Network (HAN)*, a system of neural networks that performs harmonic analysis on musical scores. Test results on 18 J.S. Bach chorales are presented. An example analysis is also shown.

1 Introduction

There has been ongoing work on the development of *Harmonic Analysis Network (HAN)*, a system of feed-forward neural networks that performs harmonic analysis on musical scores. This system takes note values as input; other information, such as note durations, key signatures, and cadence points, is not used. The output that the system gives is the complete harmonic analysis, which includes the key, root, quality, and inversion of each sonority of the piece. A sonority is defined as the set of sounding notes present whenever any note is articulated in the composition [4].

In the past, the task of computerized harmonic analysis has been researched, most recently by Taube [4] and Temperley [5]. Both of their works use an algorithmic approach. In particular, the method described in [4] was used to perform harmonic analysis on J.S. Bach chorales. There has also been work done in the past on training neural networks to harmonize J.S. Bach chorales given the melody line [3]. Given the difficulty of performing harmonic analysis using a few explicit and simple rules, we attempt to perform this task using neural networks. We chose to analyze Bach chorales due to the large sample size and their stylistic consistency.

2 Methods

The task of harmonic analysis is separated into four parts: the first part is the analysis of the key of each sonority; the second, analysis of root; third, analysis of quality; and finally, analysis of inversion. We will now describe each part in detail.

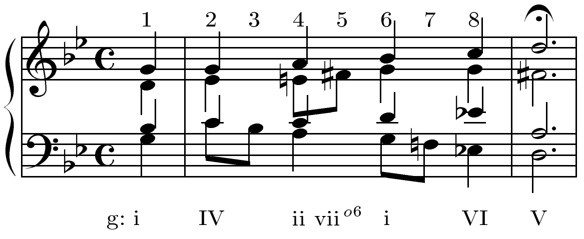

Figure 1: The opening phrase from Bach chorale "Auf Meinen lieben Gott" with analysis.

2.1 Key Analysis

This part of the analysis uses N feed-forward neural networks with one layer of hidden units (the exact value of N will be stated later). The networks in this part are collectively known as *Key Network*. The training set consists of 20 Bach chorales (10 major, 10 minor) chosen in an arbitrary way. All the chorales were taken from [1]. Each and every element in the training set corresponds to each and every sonority of the 20 chorales. Each *input* element of the training set consists of the notes (in letter names) of its corresponding sonority, as well as the notes of the n_k previous and n_k next sonorities (the relevance of this will be apparent soon). Different notes with the same letter name are ignored. Each *output* element of the training set consists of the key of its corresponding sonority.

For example, the chorale "Auf Meinen lieben Gott" was included as part of the training set, the beginning of which is shown in Figure 1.

If $n_k = 2$, then the training input corresponding to sonority #3 would be {G, Bb, D, C, E♭, G, B♭, C, E♭, G, A, C, E, A, C, F♯}. Note that the doubled

G in sonority #1, doubled C in sonority #2, and the doubled A in sonorities #3 and #4 only appear once in the training input. The training output for the same sonority would be {g-}.

To be sure, the key of any particular sonority cannot be determined using only that sonority alone. Using the example above, the notes of sonority #3 can at least belong to B♭+, E♭+, g-, or c-. But the appearance of E♮ in sonority #4 make all but g- very unlikely. Hence, it can be seen that we would need the information from the surrounding sonorities to help decide the key of the current sonority. Therefore, we have the idea of including the notes of the n_k previous and n_k subsequent sonorities in the training input, creating an "input window," $2n_k + 1$ sonorities wide, centered on the current sonority. Different sizes of windows, with $2 \leq n_k \leq 6$, were used. For each value of n_k, three networks with identical architecture were trained with different initial values; they were then joined to form a committee [2]. For any sonority, the output of the committee is simply the mode of the outputs of the members of the committee; if there is a tie, then the output is randomly chosen among the highest occurring outputs (for a committee of three, this would occur when all members gave different outputs). Since there were five different values of n_k used, there resulted five different committees (and hence giving $N = 15$). These five committees were joined together to form another committee to obtain the final output. Thus we have a "committee of committees." This idea was used because networks of different values of n_k would potentially do better than others at different passages. Networks of a smaller value of n_k would do better at passages that change keys often. Networks of a larger value of n_k would tend to "smooth" out passages: in a place where the piece is changing from one key to another, they are less likely to switch back and forth between the keys; once they decide to switch keys, they are more likely to remain there.

One detail that needs to be mentioned is that in the 20 training chorales, not all keys occur with the same frequency. If the training were done with the 20 chorales in their original keys, then it is likely that certain keys would be given undue preference and some remote keys would even be ignored. Therefore, all training chorales were transposed 12 times, such that all the chorales would be found in all keys up to 6 sharps and 6 flats. The total number of sonorities in the original 20 chorales is 1507. Therefore, the total number of elements in the training set is $1507 \times (12 + 1) = 19591$.

2.2 Root Analysis

Root Network consists of a committee of three feed-forward neural networks. The training input set consists of the notes of all sonorities of the 20 training chorales, as before, but now all elements of this set are transposed to the same reference key. So now there is only one copy of the training input set, as opposed to 13 previously. This is possible because we now have the key information for every sonority (it was provided as output of *Key Network*), and also because the output of *Root Network* is the same no matter what key the sonority is transposed into (since the root is by definition relative to the tonic of the key). The only other additional information included in the training input set is whether the sonority is in a major or minor key. This is obviously necessary since, for example, a C-major chord would be I in C+ but III in a-. The training output set consists of the root of each sonority, which is one of the following: I, II, III, IV, V, ♯VI, ♯VII, ♮VI, ♮VII. Note that *Root Network* does not attempt to distinguish between sonorities with the same root but with different qualities. This is the task of *Quality Network*, described next. Continuing with the example in Figure 1, if the reference key were a- (and window size $n_r = 2$), then the training input for sonority #3 would be {-, A, C, E, D, F, A, C, D, F, A, B, D, F♯, B, D, G♯}. The corresponding training output is {IV}. The training output is {IV} whether the corresponding training input is transposed to a-, in the original key of g-, or any other minor key.

Thus, with the knowledge of key obtained in *Key Network*, we were able to train *Root Network* with a much smaller number of training inputs. The result is faster training time. As for the window size, it is seen that for the determination of the root, the information outside the immediate vicinity of the sonority in question is irrelevant. Most of the time, only the notes of the sonority in question are sufficient to determine the root for that sonority. For example, the notes of sonority #3 are {B♭, C, E, G}, and we know the key is {g-}. Clearly, only the IV chord is suitable. However, sometimes a little bit of context is still needed. Consider this phrase from the same chorale shown in Figure 2. With no context information, the root of sonority #6 seems like a III chord. But when we look at the sonority previous to it, we realize it is just a I chord in passing. In our experi-

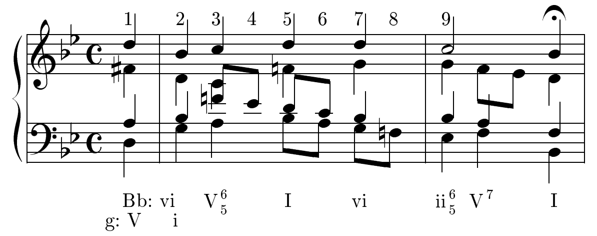

Figure 2: Another phrase from Bach chorale "Auf Meinen lieben Gott" with analysis.

ment, we used a window size $n_r = 2$. Similar to *Key Network*, we used a committee of three networks to obtain the final output for *Root Network*.

2.3 Quality Analysis

Quality Network also uses a committee of three feed-forward neural networks. The training input set is identical to that of *Root Network*, minus the mode (major or minor). The window size used is $n_q = 1$. The training output set consists of the quality of each sonority. The different types of qualities *Quality Network* attempts to classify include +, -, x (augmented), o (diminished), v7, +7, -7, o7, and ϕ7 (half-diminished 7th). Quality analysis turns out to be an even easier task than root analysis. The mode input is not necessary in this analysis because a sonority's quality is not dependent upon whether the current mode is major or minor. In Figure 2, the g-minor chord in sonority #2 is a minor chord, whether the key is g-minor or B♭-major. As with the root analysis, a little context information is necessary to provide more accuracy. Sonority #6 by itself has the quality -7; but with the context information, we can see that it is a + chord in passing.

2.4 Inversion Analysis

Inversion Analysis is by far the most simple and straightforward. It does not involve the use of neural networks because no context information is required. It just involves the use of a look-up table indexed by the key, root, and quality of the current sonority. The output is the inversion of the current sonority.

2.5 Pivot Chord Selection

The last task of *HAN* involves determining pivot chords, if and when they exist. Up until this point, only one key is determined for each sonority. A pivot chord exists when a piece is changing keys, and at the point of transition, a sonority can be found to belong to both keys. The selection algorithm is straightforward, as follows: If a passage in key X ends at sonority $\#n$ and a passage in key Y starts at sonority $\#(n+1)$, then if the notes of sonority $\#n$ belong to key Y, then the pivot chord is at sonority $\#n$; otherwise, if the notes of sonority $\#(n+1)$ belong to key X, then the pivot chord is at sonority $\#(n+1)$; otherwise, there is no pivot chord.

		no alg.		w/alg.	
Suite	# Son.	# Correct	Accu. (%)	# Correct	Accu. (%)
1	558	463	83.0	498	89.2
2	804	659	82.0	728	90.5
1+2	1362	1122	82.4	1226	90.0

Table 1: This table shows the prediction accuracy of the system of neural networks on the two suites of Bach chorales with and without the improving algorithms.

3 Results and Discussion

The system of networks was tested on two suites of Bach chorales. Suite 1 contains 8 chorales: they are also some of the chorales chosen to be analyzed in [4]. They are numbers 7, 9, 30, 50, 91, 206, 211, and 284 in the Kalmus Edition. They were chosen in [4] because they "provide particular analytical challenges." Suite 2 contains 10 other chorales which were arbitrarily chosen. They are numbers 26, 32, 57, 93, 138, 154, 175, 189, 201, and 224. The prediction accuracy of the networks on these chorales are presented in Table 1. After analyzing the mistakes the networks made, a number of algorithms were developed to improve the overall performance. A total of three algorithms were implemented, one for *Key Network*, one for *Root Network*, and one for *Quality Network*. Each algorithm takes as input the output of its associated *Network*, performs the intended function (described below), and provides the output for the next *Network* or for *Inversion Analysis*. Table 1 also shows the resulting improvement.

Briefly, the improvement algorithm for *Key Network* finds sonorities where (i) the keys determined for them are theoretically impossible; (ii) consecutive numbers of them are determined to be in the same key, yet none of them serve a dominant function in this key. When sonorities satisfying condition (i) are found, the key output is changed to that of the second-choice output of *Key Network*, if this output will negate condition (i); otherwise, it is changed to the output for an adjacent sonority, if this output will negate condition (i); otherwise, nothing is done. The second-choice output of *Key Network* is the second highest-occuring output of its final committee. When sonorities satisfying (ii) are found, the output for the sonority immediately previous to these sonorities is assigned to these sonorities. The algorithm for *Root Network* finds sonorities whose roots are completely deterministic given the key; for each, it calculates the expected root output and compares it with the actual output. If they do not match, the actual is replaced by the expected. By completely deterministic, we mean the sonority contains all and only the notes of a triad, or all the notes of a seventh chord. Granted, the roots of most sonorities are completely deterministic (79% in Suites 1 & 2), and thus the improvement algorithm, running by itself, would already produce mostly accurate results, but *Root Network* does much better just by itself (92%) and even slightly better with the improvement algorithm (93%). The algorithm for *Quality Network* performs in a similar way to that for *Root Network*.

An example analysis by *HAN* is shown in Figure 3. The analysis is mostly very well done, with a few exceptions: sonority #3 should be analyzed as g-; the pivot chord in sonority #16 should occur on the first half of the beat; sonority #40 should just be a i chord in passing. The errors at sonorities #3 and #40 represent a class of errors where the immediate context is preferred over a wider context, especially when the wider context seems strange. The problem at sonority #16 is that the pivot chord should ideally occur on the first half of the quarter note; although we do not consider this to be incorrect, the improvement can be done by considering the note durations in the pivot chord selection algorithm. By comparison, the analysis of this chorale by MTW [4] was available at the author's website. This analysis performed correctly in the aforementioned sonorities, but at sonorities #19-20, #22-23, and #27-29 it modulated briefly into C+, which was not called for. So from this example alone, it seems that both *HAN* and MTW perform well nearly all the time, but each has its own minor problems to deal with.

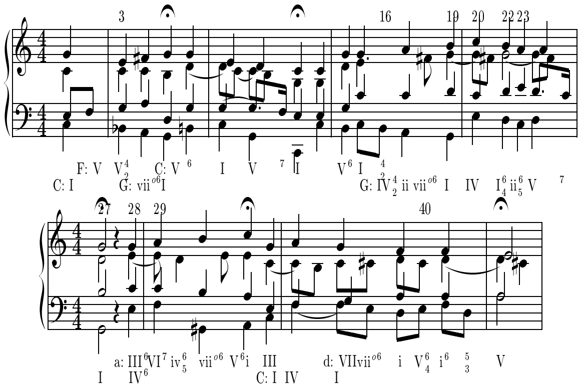
3
16
19
20
22
23
F: V V$^4_2$ C: V ^{6} I V ^{7} I V^6 I $^4_2$
C: I G: vii^{o6}I G: IV$^4_2$ ii vii^{o6} I IV I$^6_4$ ii$^6_5$ V ^{7}
27
28
29
40
a: III^{6}VI^7 iv$^6_5$ vii^{o6} V^{6}i III d: VIIvii^{o6} i V$^6_4$ i^6 $^5_3$ V
I IV^6 C: I IV I

Figure 3: Example analysis by *HAN* from Bach chorale "Ich dank' dir, Gott, für all' Wohlat".

4 Future Work

More work is needed to improve the pivot chord selection algorithm. As well, we plan to try different sizes of input windows, especially for *Key Network*. A further goal is to test *HAN* on a second set of Bach chorales. In addition, it would be interesting to retrain *HAN* on pieces by other composers and see how well it performs.

5 Acknowledgements

The authors wish to thank Mark Sallmen for verification of hand analysis of Bach chorales.

References

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

[2] Christopher Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1996.

[3] Dominik Hörnel and Wolfram Menzel. Learning musical structure and style with neural networks. *Computer Music Journal*, 22(4):44–62, 1998.

[4] Heinrich Taube. Automatic tonal analysis: Toward the implementation of a music theory workbench. *Computer Music Journal*, 23(4):18–32, 1999.

[5] David Temperley. An algorithm for harmonic analysis. *Music Perception*, 15(1):31–68, 1997.