Chapter 2 (DRAFT - 3/1/16)

Hugh Zabriskie

7 March 2016

1 Motivation for the harmonization task

The task explored in this paper is the generation of four-part counterpoint given a chorale melody. J.S. Bach's collection of over 300 chorale harmonizations is used as the dataset because it remains today one of the finest examples of four-part counterpoint. Each harmonization is a complex solution, satisfying a variety of voice leading and cadential constraints, as well as innumerable other conventions of musical style. Chorale harmonization is therefore a fundamentally difficult computational task. The reader might ask whether a rule-based approach is suitable to this task, since after all, isn't harmonization are a largely rule-based. Couldn't musical constraints would be computationally encoded and then only harmonizations that satisfy them are generated? However, a purely rule-based approach is rendered impractical by a couple important factors. One is the sheer number of conventions that would need to be encoded, each with a varying level of specificity and precedence with respect to the other encoded conventions. Determining precisely what those conventions would be is an even more difficult problem, since many chorale constraints are vaguely defined or flexible in application. Another reason for avoiding a rule-based approach is that an infinite quantity of harmonizations exist that satisfy any given chorale melody. The goal is not to simply produce satisfying harmonizations, but to *approximate* Bach's harmonization as closely as possible. In contrast, a ”learning” model updates its parameters to capture the complex correlations and patterns it discovers in the training set of harmonizations. Applying those parameters to a new chorale melody will then yield a predicted harmonization, and its predictions will directly reflect (and only reflect) the harmonizations the model was trained upon.

2 Literature Review

Hild, Feulner, and Menzel (1992) presented the first effective neural network approach for harmonizing chorales, generation harmonizations on the level of an ”improvising organist” (p. 272). The task decomposed into three subtasks, each learned by a neural net. A harmonic skeleton is first created by sweeping through the chorale melody and determining a harmony for each beat, where harmony is represented by a unique figured bass notation. For each time step t, the network takes an input a window of information, including the harmonies chosen in the interval$[t-3, t-1]$ and the melody pitches in the interval $[t-1, t+1]$. The resulting harmonic skeleton is passed to another network to generate a chord skeleton, which selects the inner voices based on the input figured bass, and a final network adds ornamental eighth notes to restore passing tones between chords. Although, HARMONET demonstrated strong success, the model used external ”chorale constraints” (ibid., p. 271) in constructing the chord skeleton in order to avoid unfavorable chord structures. The models presented in this paper attempt to learn the task of harmonization without any form of manual intervention in the network’s learning process.

Substantial work has been done in the field of music generation using RNNs and LSTMs that demonstrates their ability to learn complicated musical structures and relate temporally distant events, particularly in the realm of melodic generation. Toiviainen (1995) developed a neural network that generates a jazz bebop melody over series of chord changes. The network achieves melodic continuity by using a ”target-note technique”, where the end of a previous melody segment and the present chord are used to predict the next melodic pattern at time $t+1$, while the following chord at time $t+2$ is use to optimize of the melodic pattern, thereby smoothing the melodic transitions over chord changes. Eck and Schmidhuber (2002) improved on the results of Mozer (1994), who used RNNs to compose melodies with chordal accompaniment but found the resulting music lacked larger thematic and phrase structure. They attributed Mozer's results to the ”vanishing gradients” problem (described briefly in chapter 1) and then trained LSTMs to first generate the chordal structure and use that as a input to the LSTMs that generates a blues-style melody with promising results. Franklin (2006) also used LSTMs (two inter-recurrent networks) to compose jazz melodies over a chord progression, training the networks on a dataset of well-known jazz standard melodies.

Research specifically on models for the Bach chorales have seen a variety of approaches. Allan and Williams (2005) generated a dataset of the chorales by transposing each chorale to C major or C minor and then sampling every quarter note, now available at `http://www-etud.iro.umontreal.ca/~boulanni/icml2012`. They trained Hidden Markov Models (HMMs) on the data to model chorale harmonization, creating a probabilistic framework for deciding the most likely choice of alto, tenor, and bass notes to complement the melody in each time frame. More recent work has focused on music *generation* models rather than completive models. Boulanger-Lewandowski et. al. (2012) used this version of the dataset, along with multiple other corpuses, to make a comprehensive survey of music generation models that performed next-step prediction for all voices. Based on log-likelihood and overall accuracy on the test data, a specific flavor of RNNs was found to be most effective on all corpuses used in the study. Other studies have sought to ”re-construct” chorales using more sophisticated neural models (Liu, 2014), including a large-scale survey of polyphonic music modeling that evaluated the performance of eight LSTM variants Greff et al. (2015). Both previous papers mentioned highlight the use of RNNs and LSTMs as effective models because of their ability to learn temporal dependencies within polyphonic music (i.e. how distant musical events can be related). Notably, the latter found the ”vanilla”, unmodified LSTM equally effective as the other variants (ibid., p. 7), so this architecture was chosen for this study.

My approach consists of applying a variety of non-neural and neural models to the task of chorale harmonization. Recent harmonization and generative models (Allan and Williams, 2005; Kaliakatsos-Papakostas and Cambouropoulos, 2014; Greff et al., 2015) relied on a dataset that includes only pitch information about each time step, and the objective was in all cases next step pitch prediction. My goal was to extract a new set of features from Bach’s chorales and examine how those features improved or worsened model performance. Moreover, the objective was to learn a series of musical processes that decide the harmony at each time step that include both harmonic analysis and note selection. I chose this approach to mimic the decisions a musician would make today in harmonizing a chorale.

2.1 Harmonization tasks

In order to evaluate a model’s ability to learn different aspects of the harmonization process the decision process was broken into 4 sequential subtasks. For each melody note in each chorale, the following target outputs were extracted.

1. *Roman numeral*. The model first decided the general harmonic function by selecting a Roman numeral per Roman numeral analysis. This decision holds the most importance since it decides what notes are allowed for the alto and tenor voices.

2. *Inversion*. The chord inversion provides additional harmonic information since, for example, a I^6 harmony has different implications about future harmonies than a I chord in root position. The inversion also implies the pitch class of the bass voice.

3. *Alto*. The alto voice is then selected as the first inner voice. There is no reason for selecting the alto voice before the tenor voice since both function similarly as inner voices that support the harmony decided in the previous subtasks.

4. *Tenor*. Selection of the tenor voice completes the harmonization.

Mathematically, the algorithm should take as input the chorale, represented a sequence of notes, and it should output a corresponding sequence of 3-voice chords that represent the alto, tenor, and bass voices. We will say we have m data points with n features, and Y output classes.

- $\boldsymbol{X} \in V^{m \times n}$ is our input data
- $\boldsymbol{Y} \in Y^m$ is our output data for a given subtask
- The objective is to approximate the complex function $f : \boldsymbol{X} \rightarrow \boldsymbol{Y}$

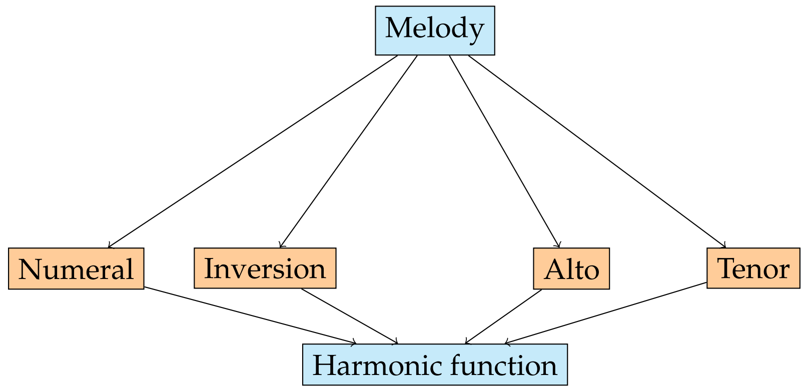
In order to generate the correct Roman numeral analysis for the first two subtasks, I relied on a combination of MUSIC21's `roman` module for initial analysis followed by substantial manual correction due some incomplete functionality in the analysis module. The alto and tenor voices were extracted as MIDI note values.

3 Methods

3.1 Important resources

We gathered chorale in MusicXML format through the MUSIC21 `corpus` module, which originally obtained the chorales from Greentree (2005). MUSIC21 is a toolkit

Figure 1: **Sequence of harmonization subtasks.**



for computational musicology maintained by Professor Michael Scott Cuthbert at MIT (Cuthbert and Ariza, 2010), and its library is built upon a comprehensive array of objects and methods that represent the fundamental musical components introduced in Chapter 1. For our purposes, music21 was used extensively to transform musical scores into symbolic representations and extracting musical features from those scores for machine learning. It was also used to generate realizations of predicted chorale harmonizations as actual scores, which are provided in Chapter 3.

In order to construct and train neural networks, the scientific computing framework Torch was used, which is written in the Lua programming language. The `rnn` library, which is designed to extend the capabilities of the `nn` module in Torch, was used to implement recurrent neural networks like RNNs and LSTM networks (Léonard, Waghmare, and Wang, 2015). Non-neural models were selected from the fully implemented classification algorithms in the machine learning library `scikit-learn` (Pedregosa et al., 2011).

3.2 Data

The dataset consists of 326 4-voice chorales gathered from the music21 library. Once collected, some manual cleaning was then performed to correct mistakes in the musicXML format related to key signatures and significant mistakes in notation (based on a visual comparison with the Riemenschneider edition of the Chorales). Chorales with significant periods of rest were removed from the dataset because 3-voice harmonies have ambiguous implications in the 4-voice model that the chorales conventionally observe. Next, the chorales were "quantized" to create strictly chordal progressions of 4-voice harmony. Like modern church hymns, Bach's chorales are uniformly structured as chordal progressions, with a consistent beat-long rate of harmonic transition. Therefore, the process of quantizing each chorale into a series of discrete and uniform time steps, each the length of a beat, could be accomplished without damaging the underlying harmonic progression. Eighth notes were removed from the voices to eliminate additional rhythmic complexity. Eighth notes facilitate voice leading in Bach's harmonizations, but rarely function as an essential part of the harmony, making their removal a reasonable design decision. This process of quantizing the chorale into quarter-note samples has been found to be effective in several other studies (Hild, Feulner, and Menzel, 1992; Madsen and Jorgensen, 2002; Kaliakatsos-Papakostas and Cambouropoulos, 2014).

3.3 Feature Extraction

In order to learn each of these subtasks, the following information was extracted for each melody note in each chorale.

1. The number of sharps in the key signature. Flats were given negative values.
2. The mode (i.e. major or minor) of the chorale.
3. The time signature of the chorale.
4. Beat strength, or metrical accent. A 4/4 measure would be assigned the following pattern: [1.0, 0.25, 0.5, 0.25]
5. The presence of a fermata - a binary feature indicating a cadence.

6. Number of beats until the next fermata.
7. Number of measures until the end of the chorale.
8. The melody pitch, encoded as a MIDI value. A search across all chorales indicated that each voice had a well-defined pitch range, verified by Madsen and Jorgensen (2002):
    - *soprano*: [60, 81]
    - *alto*: [53, 74]
    - *tenor*: [48, 69]
    - *bass*: [36, 64]
9. The interval to the previous melody note.
10. The interval to the next melody note.
11. The Roman numeral for the previous time step.
12. The inversion for the previous time step.

Features 11 and 12 were used only for Oracle experiments and demonstrating the potential of RNNs, discussed later in Chapter 3.

The Python script used to preprocess the chorales (see wrangle.py in Appendix B) [1] extracts the above features from each time step of each chorale, and stores the generated training and tests in an HDF5 file. For baseline models, the extracted data was fed as input to another Python script that performed classified learning using models from theSCIKIT-LEARN module. For neural models, the data was fed into Lua scripts that use the scientific computing framework Torch to construct models and training on supervised classification tasks.

[1] There will be an Appendix B that contains the most important code for the project.

4 Baseline models

Due to unique classification problems chosen for this paper, previous computational research on the chorales does not provide any adequate baseline models. Baseline models are important because they provide a basis for comparison when evaluating other models. As the complexity of the model changes and the features are added or removed, it is important to have a metric to compare against to see how those changes improved or worsened the results. In classification problems, a crude baseline model can be achieved by choosing the class with the most observations and using that class as the result for all predictions. TABLE 2 lists the baseline frequencies for the most common classes for each subtask. I also trained three other classifiers to get a sense of the complexity of each subtask. These classifiers are described below:

Multiclass logistic regression was introduced in Chapter 1 as a generalization of the binary classification system of logistic regression. Despite its name, this regression is a linear model. The objective is to minimize the following cost function, given the learned parameters θ.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\{\boldsymbol{y}^{(i)} = k\}\log\left(\frac{\exp(a_{ik})}{\sum_j \exp(a_{ij})}\right)$$

Where m is the number of examples, k is the number of classes, and a_{ik} is the "activation" function $\theta^{(k)T}x^{(i)}$, denoted for the ith example and the kth class.

Multinomial naive Bayes generalizes the naive Bayes algorithm for multi-class data, and it makes the "naive" assumption of independence between every pair of input features. The assumption therefore states that, given the input vector $\boldsymbol{x}$ and the class $c \in [1, K]$

$$P(\boldsymbol{x}|c) = P(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n|c) = \prod_{i=1}^{n} P(\boldsymbol{x}_i|c)$$

And based on that assumption, this baseline classifier the predicted output class is decided by

$$P(c|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|c)P(c)}{\sum_j P(\boldsymbol{x}|C_j)P(C_j)} = \frac{\prod_i P(\boldsymbol{x}_i|c)P(c)}{\sum_j \prod_i P(\boldsymbol{x}_i|C_j)P(C_j)}$$

Random forests are a powerful supervised learning technique that involves classification based on a the majority vote of a series of decision trees. Each tree is initialized with data from a random subset of features and then is trained on the data by sampling with replacement. This randomness is known to be highly effective in preventing overfitting on training data, and random forests generalize well on weaker datasets where one or more training examples do not strongly suggest differences between classes (Breiman, 2001, p. 18).

The results for each baseline model are described in Table 1.

Table 1: **Baseline model test accuracy on harmonization subtasks.**

Classifier	Numeral	Inversion	Alto	Tenor
Multi-Class Logistic	31.61%	59.76%	37.55%	37.86%
Multinomial Naive Bayes	27.44%	56.66%	35.06%	34.40%
Random Forests	49.29%	61.64%	49.44%	45.43%

Table 2: **Most common class frequency.**

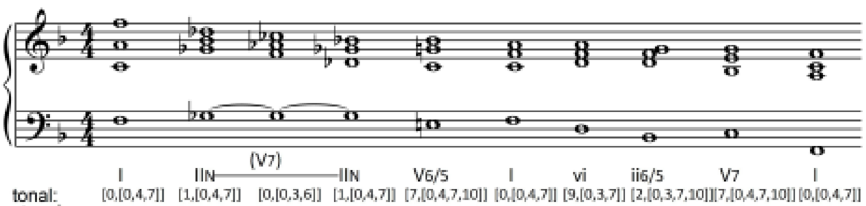
Subtask	Training Set	Test Set
Numeral	19.2%	19.2%
Inversion	55.5%	57.6%
Alto	15.7%	14.6%
Tenor	15.6%	15.5%

4.1 GCT Algorithm

While Roman numeral analysis has been the traditional method for describing harmony in the Chorales, it presents issues for statistical learning. Roman numeral classification mainly depends on the key signature, but also requires the context of the preceding harmonies. For example, a D major chord in a C major chorale might be labelled as a II or V/V depending on whether a modulation to D major had occurred or whether the preceding harmonies indicate that it functions as a secondary dominant. During training, finding two or more inputs that suggest a D major chord but are labeled differently can cause confusion in learning, particularly since in non-recurrent models there is no sense of context about other local harmonies. Roman numeral chord labeling can be further complicated by the presence of non-chord tones, which makes, for example, differentiating IV^6 and ii^7 chords computationally difficult.

The general chord type (GCT) representation provides an idiom-independent solution to encoding harmony that assigns a unique encoding to each chord, regardless of context (Cambouropoulos, Kaliakatsos-Papakostas, and Tsougras, 2014). To encode a chordal harmony, the GCT algorithm takes as input the chord to be encoded, a pitch scale that describes the tonality, and a binary ”consonance vector” $\boldsymbol{v}$ such that $\boldsymbol{v}[n] = 1$ if an interval of n semitones is considered consonant for $0 \leq n \leq 11$. In this study, I chose $\boldsymbol{v} = [1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0]$. GCT then constructs an ordering of the chord pitches that maximizes the consonant intervals between all pairs of pitches. The remaining notes that create dissonant intervals are labeled as ”extensions”. The algorithm outputs a encoding of the form [root, [base, extensions]], where root is the pitch class of the chord root relative to the tonic, and the base is the ordering of maximal consonance. I adapted the algorithm to also output the degree of inversion, where 0 represents root position, 1 represents first inversion, and so on. Figure 2 demonstrates an application of the GCT algorithm to a tonal harmonic progression, comparing the Roman numeral analysis with the GCT encoding. The base [0, 4, 7] encodes a major triad, while [0, 3, 7, 10] encodes a minor seventh chord.

Figure 2: Example of the GCT and Roman numeral notation.



In comparison with a Roman numeral analysis dataset compiled by David Temperley, GCT labeled 92% of the chords accurately, of which about 1/3 of mislabeled chords were diminished sevenths - excusable because each note in the chord can function as the root (ibid.). In order to minimize duplicate encodings, I implemented the following policies, partially drawn from suggestions by the original authors.

1. For dyads, prefer an interval of a 5th over a 4th, and an interval of a 7th over a 2nd.

2. Preference encodings where all intervals are larger than a major 2nd. This heuristic preferences a minor 7th or a major chord with an added 6th, and generally more evenly spaced encodings.

3. If more than one encoding remains, choose randomly.

The GCT algorithm was incorporated into a newly generated dataset for the chorales by replacing the numeral and inversion subtasks in $\boldsymbol{Y}$ with the new root, base, and inversion subtasks. The root subtasks establishes the chord root, while the remaining chord structure is decided by the base. As in Roman numeral analysis, the inversion implies which chord tone is assigned to the bass. As well, the new dataset classified the tenor and alto voices by their distance from the tonic pitch, instead of encoding MIDI values directly, in order to make the decision key-independent and reduce the output class space to a maximum of 12 classes (for the 12 chromatic intervals).

The same baseline models were used to evaluate the new subtasks. In both cases, there were 293 chorales in the training set, and 33 chorales in the test set.

Table 3: **Baseline model test accuracy with new GCT subtasks.**

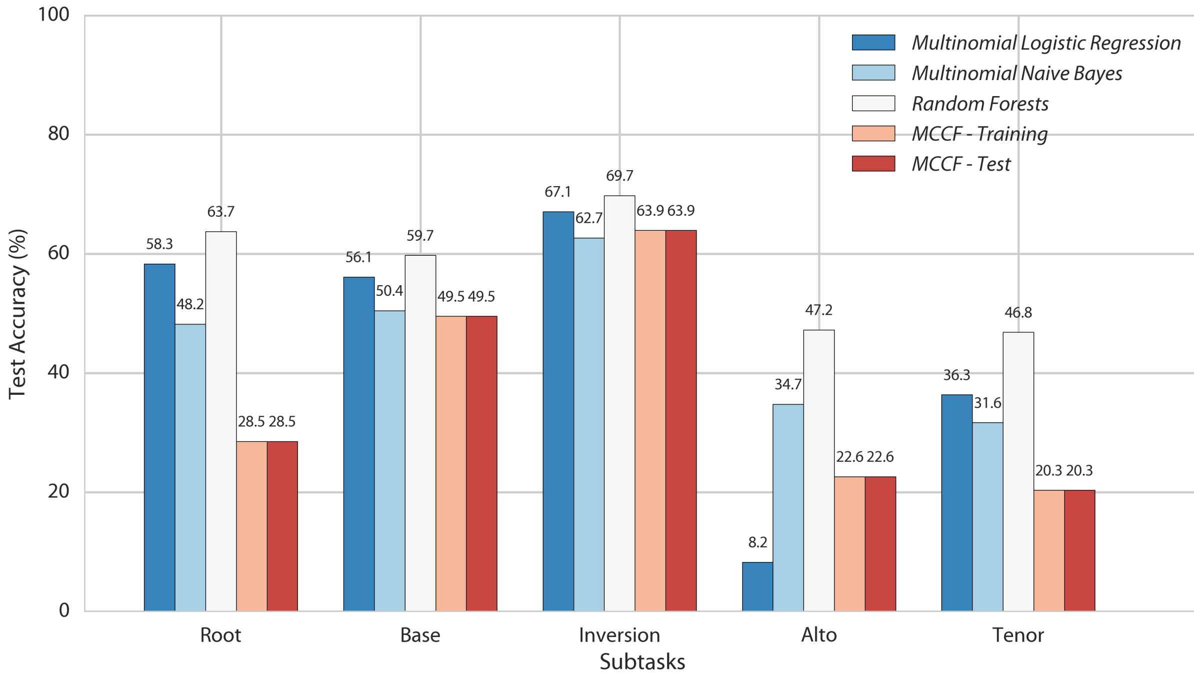
Classifier	Root	Base	Inversion	Alto	Tenor
Multi-Class Logistic	58.29%	56.09%	67.06%	38.22%	36.33%
Multinomial Naive Bayes	48.19%	50.43%	62.66%	34.73%	31.65%
Random Forests	63.71%	59.72%	69.73%	47.22%	46.81%

Table 4: **Majority class frequency (MCCF), with GCT subtasks.**

Subtask	Training Set	Test Set
Root	28.5%	28.0%
Base	49.5%	49.9%
Inversion	63.9%	65.5%
Alto	22.6%	23.7%
Tenor	20.3%	22.0%

Figure 3 provides a visual comparison of baseline model performance across all harmonization subtasks. For all subtasks, a majority of the multinomial models outperformed the MCCF baselines. In particular, Random Forests consistently outperformed the other models, with a 35% increase in accuracy over the MCCF baseline for the root subtask. However, the multinomial models struggled to perform above the MCCF baselines for both the base and inversion subtasks, which appears correlated with a high predominance of a single class in the dataset. This points to an issue with an imbalance class distribution in the data. Of the 133 output classes for the GCT base subtask, the most common class is associated with 50% of all observations in entire dataset (this happens to be the major triad), and

Figure 3: Harmonization subtask accuracy comparison



the top 3 most frequent classes account for 77% of all observations. Consequently, the vast majority of output classes are observed too infrequently in the training data to be classified accurately in the test data. Imbalanced data is a widely recognized phenomenon in data mining that is comprehensively detailed in Sun, Wong, and Kamel (2009). The significant advantage achieved using random forests can potentially be explained by its known effectiveness at classification when one or more observations is not sufficient to generally distinguish a class (Breiman, 2001, pg. 18).

References

Allan, Moray and Christopher KI Williams (2005). "Harmonising chorales by probabilistic inference". In: *Advances in neural information processing systems* 17, pp. 25–32.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Cambouropoulos, Emilios, Maximos Kaliakatsos-Papakostas, and Costas Tsougras (2014). *An idiom-independent representation of chords for computational music analysis and generation*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Cuthbert, Michael Scott and Christopher Ariza (2010). "music21: A toolkit for computer-aided musicology and symbolic music data". In: *International Society for Music Information Retrieval*, pp. 637–642.

Eck, Douglas and Juergen Schmidhuber (2002). "A first look at music composition using lstm recurrent neural networks". In: *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*.

Franklin, Judy A (2006). "Jazz melody generation using recurrent networks and reinforcement learning". In: *International Journal on Artificial Intelligence Tools* 15.04, pp. 623–650.

Greentree, Margaret, ed. (2005). *Chorales harmonized by J.S. Bach*. Retrieved October 2015 from music21.

Greff, Klaus et al. (2015). "LSTM: A Search Space Odyssey". In: *arXiv:1503.04069*. URL: `http://adsabs.harvard.edu/abs/2015arXiv150304069G`.

Hild, Hermann, Johannes Feulner, and Wolfram Menzel (1992). "HARMONET: A neural net for harmonizing chorales in the style of JS Bach". In: *Advances in Neural Information Processing Systems*, pp. 267–274.

Kaliakatsos-Papakostas, Maximos and Emilios Cambouropoulos (2014). "Probabilistic harmonization with fixed intermediate chord constraints". In: *ICMC—SMC*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Léonard, Nicholas, Sagar Waghmare, and Yang Wang (2015). "rnn: Recurrent Library for Torch". In: *arXiv* 1511.07889.

Madsen, Soren Tjagvad and Martin Elmer Jorgensen (2002). "Harmonisation of Bach chorales: KBS project report". In: p. 31.

Mozer, Michael C (1994). "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing". In: *Connection Science* 6.2-3, pp. 247–280.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Sun, Yanmin, Andrew KC Wong, and Mohamed S Kamel (2009). "Classification of imbalanced data: A review". In: *International Journal of Pattern Recognition and Artificial Intelligence* 23.04, pp. 687–719.

Toiviainen, Petri (1995). "Modeling the Target-Note Technique of Bebop-Style Jazz Improvisation: An Artificial Neural Network Approach". English. In: *Music Perception: An Interdisciplinary Journal* 12.4, pp. 399–413. ISSN: 07307829. URL: http://www.jstor.org/stable/40285674.