

Chordini

Using Recurrent Neural Networks With BLSTMs To Create and Explore Chord Progressions

Elliot Chuh

Abstract

Music generation using Machine learning and Neural Networks generally follows the same practice as text generation or Natural Language Processing. The parallels lie in nearly every aspect where each letter can be considered as a note, each word a chord, and each sentence a progression. Every phrase can be written in a way that incites a different emotion depending on the order in which each unique word or chord leads to another.

In language, a thesaurus can be used to expand the writer's vocabulary to add new flavors in delivery. The goal of Chordini is to generate a musical chord progression but to also supply a thesaurus of chords for musicians to explore new chords.

It's no secret that music can be generated with neural networks but aside from hooktheory.com, visual exploration of music is not a widespread practice. Chord progressions are also overlooked for the fancier melody generators based on musical genre.

Process

To accomplish these objectives, a recurrent neural network will be employed to accept a chord or sequence of chords and supply an output that continues or completes a musical idea based on its training data. Both objectives use the same data sources to both train and mine for pertinent information. The details of the data will be explained later on but the plan for data processing is to break down a vast variety of musical pieces to extract their chord progressions along with each chord's attributes. Chord progressions will be organized into a dataset used for the neural network while the chord contents will be used for data mining, clustering efforts, and final digital-musical translation to create an audible output.

Setup, requirements, and execution instructions are detailed in the README.txt file.

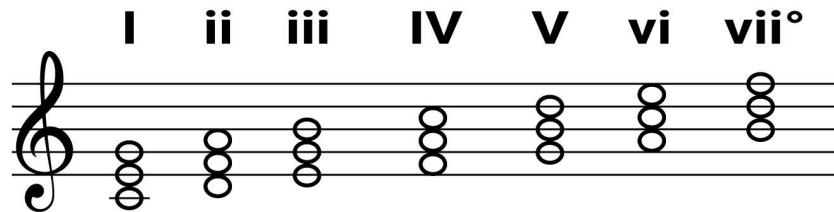
Data

Source data for this project uses only .mxl formats of musical scores that were obtained from a Musenscore forum post found [here](#). Musenscore is a musical application that is used by musicians to create musical scores and in this case, lead sheets. Lead sheets differ in traditional musical scores by generally including chord notations over melodies. This makes .mxl files more advantageous than MIDI files, which would require the "squeezing" of notes to determine the underlying chord in a musical phrase that includes both melody and moving bass lines. These .mxl files can be directly viewed with the Musenscore application which comes with the music21 library installation.

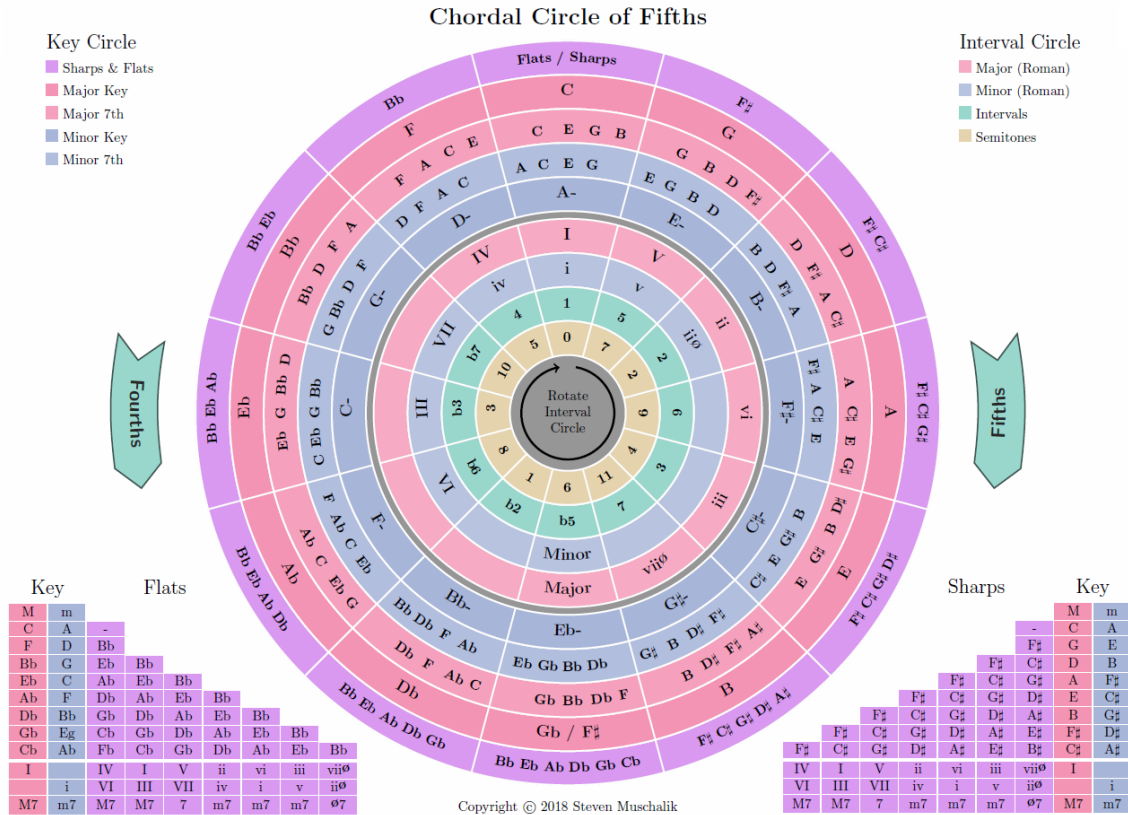
Data Extraction

Using the music21 library, created by Michael Scott Cuthbert from MIT, the .mxl data can be extracted without having to read through every .mxl file. This portion of the project is done through a python script `chord_extraction.py` where each .mxl file is parsed through to find musical events in sequence. If an event is a chord object, then that chord is recorded to build a progression while the chord contents are saved separately into dictionaries and .json files that contain the chord's name, pitches as well as the chord's roman numeral notation. This is the point when some music theory knowledge is required to understand the wide variability in what differentiates one chord from another and thus being able to simplify our data to achieve a very specific goal.

1. Major 2. Minor 3. Minor 4. Major 5. Major 6. Minor 7. Diminished



The simplification lies in the concept of musical transposition where all the songs are now pertaining to the key of C. This can be considered as the human language equivalent of using English compared to all languages and dialects.












One key aspect to note during chord extraction is the decision to remove all consecutive duplicates of a chord. This means that though a song might have a certain chord repeat for N events, only the first occurrence is recorded so that no two chords in sequence are the same. Since the goal is to help output an outline for a song via a chord progression, consecutively repeated chords are not needed. As with natural language, a word is seldomly repeated unless conveying an emphasis such as a description for a song being “really really good”.

Data Analysis (Chordini_NB.ipynb - Jupyter Notebook following chord_extraction.py)

Progressions: Recurrent Neural Network

Upon importing the songs_df.csv as a pandas DataFrame, analysis involves recording the number of unique chords in each song as well as the length of the song, calculated by how many chords exist. This sets up the ability to determine what songs are too short and what songs lack chord variability. Chord variability is restricted to be more than one standard deviation below the mean (songs need more than 8 unique chords). Then, both the progression data and the chord content data go through the same process of finding chords with the same pitch content to consolidate chord names. This needs to be addressed otherwise some chords will be recorded as occurring in only a particular song. The variation in chord name stems from the .mxl file after completing transposition to the key of C. Here is a chart that shows some of the complexity involved when naming a single chord.

<i>Chord Type</i>	<i>Symbols Used</i>	<i>Notes Included</i>
Major Triad	C	
Minor Triad	Cm, C-, Cmi, Cmin	
Diminished Triad	C ^o , Cdim	
Augmented Triad	C ⁺ , Caug, C ^(#5)	
Minor Seventh	Cm ⁷ , C- ⁷ , Cmi ⁷ , Cmin ⁷	
Dominant Seventh	C ⁷	
Major Seventh	Cmaj ⁷ , CΔ ⁷ , Cma ⁷ , CM ⁷	
Fully Diminished Seventh	C ^{o7} , Cdim ⁷	
Half Diminished Seventh	Cm ^{7(b5)} , C ^{o7} , C- ^{7(b5)}	

Chord Content Data: Data Mining and Clustering

_____ Using the chord_dict.json and the roman_dict.json files created from chord_extraction.py, the first step is to create a pandas DataFrame using both dictionaries. As with the progression data, chord consolidation is done with the chord content data. From there, various elements can be separated into their own columns. Most notably, the roman numeral interval is extracted from the column so as to only maintain the number 1-7. Upper and lower cases of the roman numeral notation signify whether this chord is a major or minor chord and thus also extracted into its own column. From the chord name, added features are extracted that include any additional intervals, denoted by numbers or transformations such as the “diminished” or “sustained” variation. Accidentals, such as sharps and flats are not maintained for the sake of simplicity and words like ‘alter’ are also excluded. The final two actions that need to be completed, pertain to the simplification of the notes that make up each chord.

A specific note can exist in every octave and this is denoted by the number attached to the note (A1, A2, A3, etc). The numbers are dropped and thus leads to more duplicated chords. However, this is not addressed because by name, these chords are technically different (research concept of “slash” chords). The other cleanup item required surrounds the notion that a single note can be written in multiple ways with the use of “accidentals”. As noted above, accidentals are denoted with sharps(#) and flats(b) and these signify a half-step change in a higher and lower direction, respectively.

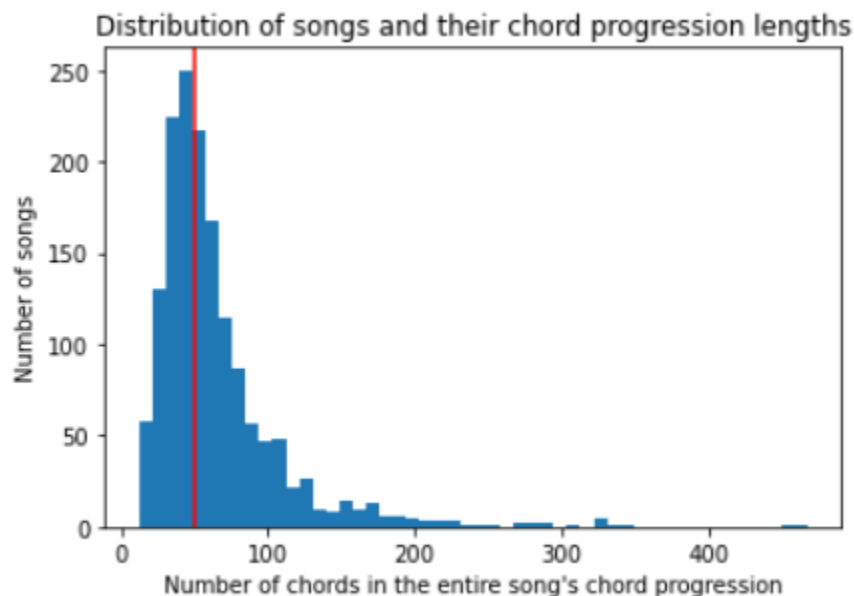
	D \flat	E \flat		G \flat	A \flat	B \flat		D \flat	E \flat		G \flat	A \flat	B \flat		
	C \sharp	D \sharp		F \sharp	G \sharp	A \sharp		C \sharp	D \sharp		F \sharp	G \sharp	A \sharp		
C	D	E	F	G	A	B	C	D	E	F	G	A	B	C	

A B \flat is the same as an A \sharp and a C \sharp is the same as a B $\sharp\sharp$. A dictionary is used to force all variations to a single arbitrarily defined notation.

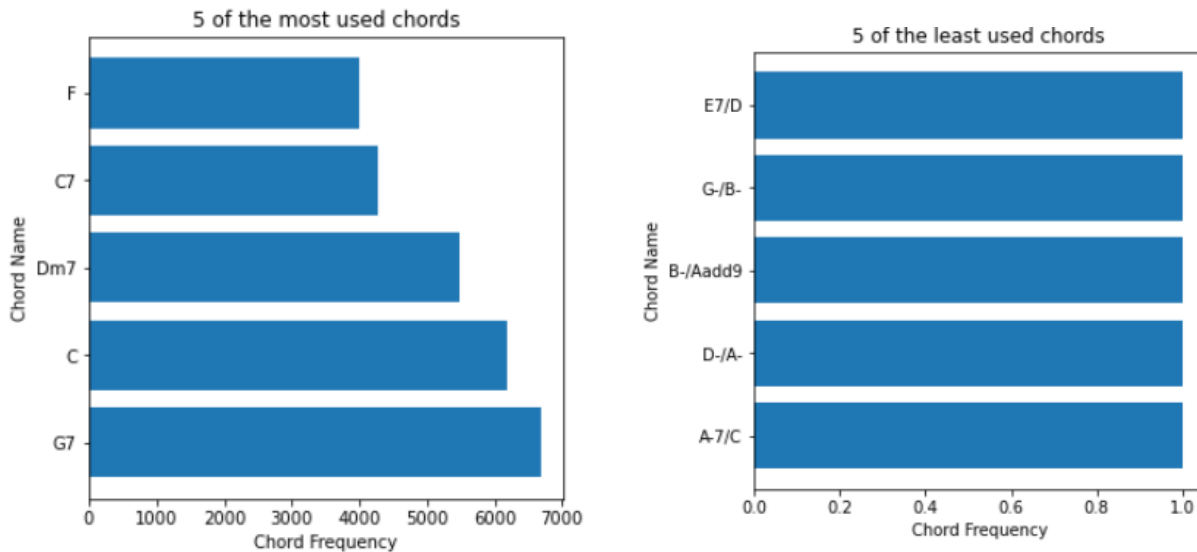
PreProcessing

Progressions: Recurrent Neural Network

Because the RNN requires the same input sequence length across all samples, preprocessing requires this step along with chord-to-number tokenization. This is achieved by using Keras' tokenizer and zero padder. Based on the distribution of chord progression lengths in the sample data, a sequence length of 50 is chosen as the cutoff. This means that all song data beyond the 50th chord will not be used. With the repetition aspect of songs, this does not affect the information retention within the range of the first 50 chords. As part of this 50-chord cutoff, any songs that have less than 50 chords are pre-padded with zeros. Further study into the sequence length will be covered as it is not the intention to feed sequences of length 50 into the RNN.



Once tokenization is done, further insights can be gathered in terms of the frequency of chord appearances. Here are the top 5 and bottom 5 chords. Keep in mind that there are 1221 chords in between these results.



With any RNN model comes the concept of a “sliding window” to determine how the neural network will train. At the moment, it is undesirable for the network to predict just the final chord based on the 49 chords that preceded it. There is no chance that another song will have the same preceding 49 chords. Training and validation accuracies start further from each other as the sequence length is shorter. If the sequence length is set to 10, the training accuracy would start at ~80% while the validation accuracy starts ~30%. At longer lengths, they at least start together so that a diverging point can be observed. A length of 25 was decided on as was half the length of the entire song’s length. After a train test split of 80/20, preprocessing is complete and the data is ready to be used for training.

Chord Content Data: Data Mining and Clustering

The chord content data with the added categorical features is used as-is for data mining and visualizations. For clustering models, the data simply progresses through one-hot encoding using Sklearn’s multi-label binarizer. Once select columns are dropped to combat collinearity, the dataset is ready for clustering models. At this point, scaling can be performed but is unnecessary because all features are categorical and hold either 0 or 1. Normally clustering is not advantageous for categorical features but can still be leveraged considering the subjective nature of music. Also, it can be used to compare the difference between K Means and K Modes. K Modes is a model created by Zhexue Huang in 1998 and can be studied in this [paper](#). Where K Means uses distance to measure clusters, K Modes quantitatively calculates the dissimilar characteristics between two objects. This is the reason for excluding scaling efforts as a whole. Disregarding scaling discourages the use of PCA which is irrelevant in this effort since the aim is to keep as much data that defines a chord.

Modeling

RNN

Multiple architectures and features were tested however, results did not exceed 40% accuracy. The RNN goal was split into 2 use cases for the output. One employs a many-to-one

methodology that outputs a chord progression that continues from an input chord progression given after also providing an output chord progression length. This output is predicted based on an input parameter to a function that determines whether the best or worst probabilities should be chosen. Another function that employs a one-to-one methodology was written to accept one chord and output the best 15 probable chords. The uses of these outputs will be discussed in the results section. Here are some features that were explored during the testing of different RNN architectures.

- *Embedding Layer*

The purpose of the embedding layer is to add dimensionality to a label encoded list of tokens. Since each chord is represented by an integer, the model has no way to define these chords by some set of attributes and so an embedding layer allows chords to be differentiated in more ways than just the identifying integer. An embedding dimension of 24 was chosen to attempt to mimic the 12 intervals between octaves in addition to 7 different modal scales and 5 common variations of chords (major, minor, diminished, augmented, slash)

- *Bidirectional Long Short Term Memory*

The bidirectional LSTM cell provided the best results as bi-directionality allows the neural network to keep all chord transitions within the context of the preceding chords. The LSTM also allows for a longer context reference compared to the SimpleRNN as longer phrases are remembered in the “long-short-term” sense. The LSTM is more advantageous to the SimpleRNN due to the memory handling nature that addresses the issue of the vanishing gradient as sequences get longer. This is due to the fact that backpropagation experiences a negative effect as small weights are multiplied with even smaller weights.

- *Deep Bidirectional LSTM*

A deep neural network opened doors to the implementation of other features such as the autoencoder as well as network symmetry and perceptron complexity. Results were very similar to the single-layer BLSTM for reasons that will be explained in the results. Simpler models opted for computational ease.

- *Gated Recurrent Unit*

The GRU differs from the LSTM by lacking an output gate and additional parameters during cell configuration. GRU cells are known to handle infrequent datasets well however did not perform noticeably better than the LSTM in terms of test accuracy. Performance floated around ~35%.

- *Autoencoder*

The autoencoder also did not add gains to accuracy when implemented with the deep network architecture. The use of the autoencoder was intended to maximize the use of the LSTM cell’s hidden state by obtaining the return sequence and repeating vector calculations. Again, simpler models were opted for computational ease.

- *Time distribution*

Time distribution was also tested only to find the irrelevance given the purpose of the model. Utilization of time distribution would assume that a many-to-many problem is at hand. The output would then change according to the accumulation of hidden states as the training progressed through multiple timesteps within the N-length sequence provided. Since the BLSTM sufficed in predicting a chord given the context of the N preceding chords, the separate timesteps of the output did not have to depend on the sub-length timesteps the preceded it. To clarify, the 3rd chord in the output sequence did not depend on the first two time steps of the input sequence.

The resulting BLSTM RNN had the following configuration:

- **1 embedding layer** of 24 dimensions
- **2 hidden layers** consisting of BLSTMs with 128 and 256 nodes. The 'relu' or 'rectifier' activation function is used to capture only the positive output. These layers included batch normalization and dropout layers.
- **1 output dense layer** with a softmax function to support the multi-classification aspect of the problem.
- **Loss** was calculated using sparse categorical cross entropy to address the multi-class aspect of token labels.
- **The Adam optimizer** was used with a learning rate of 0.001. Results did not benefit changes between .0001 and .005, however larger learning rates reduced accuracy below 20%.
- **Batch size** = 64
- **Epochs** = 75

K Means / K Modes

Both models were trained but only results from the K Means model were used to output any file and will be discussed in the results.

- *K Means*
K Means clusters were determined by first running a loop to find an optimal cluster number by observing the results of a scree plot and a silhouette plot.
- *K Modes*
K Modes has 2 modes - Cao and Huang. Both were similar and resulted in similar loss results which are relative to the data similar to K means inertia. The 'elbow' of the plot reveals the target number of clusters to use. Further results were not processed as was done for K Means and will not be discussed in the results.

Results

RNN

- Though the train accuracy reached as high as 90%, the validation scores did not exceed 40%. This is likely attributed to the fact that the final dataset had a large amount of class

imbalance. Also repetition in music might amplify the effect of a wrong prediction by the factor of the repetition.

- As mentioned previously, the RNN model results were used in a many-to-one function that takes an input chord progression sequence, a target number of chords to output, and a flag to determine whether the best or worst options are desired. This output is then used as an input to the `prog_player` script that converts the chords to create a score in Musescore. Then the audio can be saved as an .mp3 file.

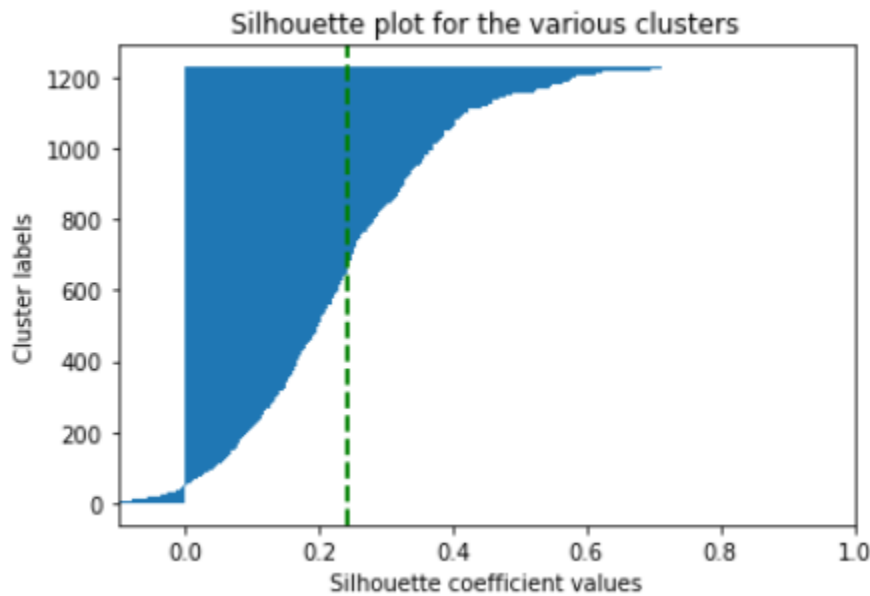
Another function was written to accept one chord and output 15 of the best options. This output was then regenerated to include the input chord before each of the potential 15 output options. When converted into Musescore, the audio is meant to be able to provide the user with the effect of each transition from the input chord to one of each of the 15 options.

- These functions can be rewritten to access the chords with intermediate probabilities, providing more interesting chord options depending on the variability desired.

K Means

- The following undesirable results are proof that clustering of categorical features are not ideal. The silhouette plot below shows that nearly half of the chords did not neighbor a

cluster center within the average silhouette score.

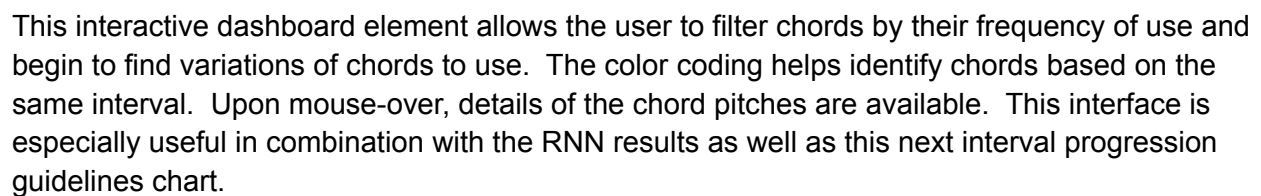


However, interesting insight was still obtained by retrieving the chords of the 85 cluster centers. This output file can also be converted into musical score and .mp3 with the music21 library and Musescore. The audio file km_cluster_center_chords.mp3 can be used to find chords that are so dissimilar that they create very interesting options for jumping from one chord to another. This could prove to be a challenging musical exercise for a musician to create a melody that transitions across a selection of these chords.

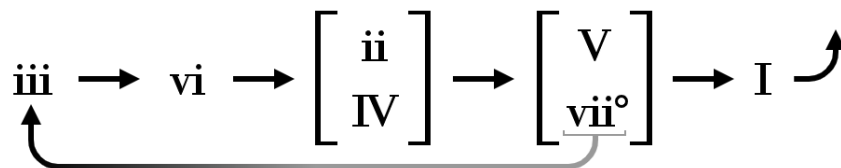
Cluster centers @ 85 clusters:

Am11C7 E7 Cm7 Dm9/G D A-7 B-add9 D- B-m G7 E- E13 Dm6/B F#7 A7 Fmaj7 C-/G-
 B/o7 D/o7 Fadd9 Gm A-m B Em D#dim/B# D9 G9 A9 Fm F7 A-9 B7 Am7 D13 Cadd9
 D-maj7 G- D-9 E-9 Am/F# B-maj7 E-m Cm D-6 Bdim/G# F/A F#13 D#7alter#3 B-6add9
 B13 F#sus Bm E/o7 Dm Edim/C# F#m G/B D#13 AmM7 D-- D-7addb9 A- C6/D D-m7
 Em9 C#7alterb5 Dm7 Cmaj7 D7 F- A6 D7alter#5addb9 B- G-9 Gm7 A B# E-7 B-7
 A-maj7 CM13 D+ D-13 Fm7

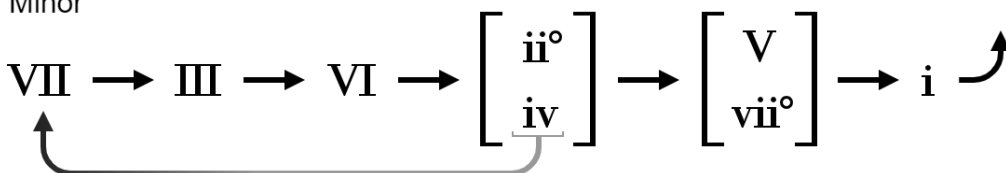
Chords in key of C by frequency and features



Major



Minor



This chart is a general map of what chord intervals transition especially well. By knowing what interval the user wants to land in, they can filter for chord options using the tableau interactive visualization.

Another useful application in conjunction with the interval map above is provided below.

Chords in key of C by interval and features

Roman Interval List														Roman Interval List	
['I']	['I']	['ii']	['ii']	['iii']	['iii']	['iv']	['iv']	['V']	['v']	['VI']	['vi']	['VII']	['vii']	<input checked="" type="checkbox"/> (All)	
C#6/G#..	Cm6ad..	D-m6	D6add9	E6add9	E-m6	Fm6add9	F#6add9	G6add9	G-m6	A6add9	A#m6a..	B6add9	B-m6	<input checked="" type="checkbox"/> ['I']	
C-6	C-m6	D-m6/G	D-6add9	E-6add9	E-m6	F#m6	F#6	G-6add9	G-m6	A-6add9	Am6/F#..	B-6/Cad..	B-n	<input checked="" type="checkbox"/> ['I']	
C#6	C#m6	D#m6/..	D-6add9	E#6/G#..	E-m6/G-	Fm6	F7addb..	G6	G-m6/B-	A#6add9	Am6ad..	B-6/Fad..	B-n	<input checked="" type="checkbox"/> ['ii']	
C7addb..	C#m6/E	D-m7	D#6add9	E6	Em6	Fm6/D	F7addb9	G6/D	G#m6	A6	A-m6	B-6add9	B-m6	<input checked="" type="checkbox"/> ['iii']	
C7/D-ad..	Cm6	D-/o7	D6	E6/B	Em6/B	Fm6/G	F7addb9	G-6	Gm6	A6/C#	A-m6	B--6add9	B-m6	<input checked="" type="checkbox"/> ['iv']	
C7/Ealt..	Cm6/D#	D-m7	D6/A	E-6	Em6/C#	Fm6/G#	F-7add..	G-6	Gm6/A	A-6	A-m6/F	B6	B-m6	<input checked="" type="checkbox"/> ['v']	
C7+add..	C-/o7	D-m7/F-	D6/E	E-6/G	E-/o7	F-/o7	F-7alter..	G6/D	G#m6	A-6/B-	Am6	B-6	B-m6	<input checked="" type="checkbox"/> ['vi']	
C7add#9	C-m7	D-mM7	D6/F#	E#6	E-m7	F-m7	F-7alter..	G7addb..	G-m7	A-6/C	Am6/B	B-6/C	Bm	<input checked="" type="checkbox"/> ['vii']	
C7addb9	C-m7/G-	D/o7	D-6	E7/Bad..	E-m7/D..	F/o7	F#7add..	G7addb..	G-/o7	A-6	Am6/C	B-6/D	B-n	<input checked="" type="checkbox"/> ['vii']	
C7add..	C#m7	D-/o7/A-	D-6/C	E7/F#..	E-/o7	F-/o7/B	F#7add..	G7/Cad..	G-m7	A7addb..	Am6/E	B-6/F	B-n	<input checked="" type="checkbox"/> ['vii']	
C-7add..	C#m7	D-/o7/C	D-6	E7/F#..	E-m7	F#m7	F7add..	G7/G#a..	G/o7	A7add#..	Am6/F#	B-6	B-/c	<input checked="" type="checkbox"/> ['vii']	
C7/F#..	C#m7/G	D/o7/G	D#6	E7add#9	E-m7/A-	F#m7	F#7add..	G7addb9	G/o7/C	A7/B-ad..	Am6/G	B#6	B-/o7	<input checked="" type="checkbox"/> ['vii']	
C#7add..	C#m7	D-/o7/G	D7addb..	E7addb9	E-m7/D-	F#m7	F7addb..	G7addb9	G#m7	A7/C#a..	AmM7a..	B-7add..	B-n	<input checked="" type="checkbox"/> ['vii']	
C#7add..	C#m7/..	D-/o7/G#	D7/E-ad..	E-7add..	E-m7	F#m7	F#7add..	G7alter..	G#m7	A7/Ead..	A-m7	B7/C#..	B-m7	<input checked="" type="checkbox"/> ['vii']	
C7/B-ad..	C#m7	D#m7	D7add#9	E-7add..	E/o7	F#m7/E	F7	G-7add..	G#m7/..	A7add#9	A-/o7	B7/Cad..	B-m7	<input checked="" type="checkbox"/> ['vii']	
C7add..	C#m7/B	D#m7	D7addb9	E-7add..	E/o7/A	F#m7	F7/A	G#7/Aa..	Gm7/A	A7addb9	A-m7	B7/E#a..	B-m7	<input checked="" type="checkbox"/> ['vii']	
C-7add..	C#m7	D#m7/..	D7alter..	E#7add..	E/o7/B-	Fm7	F7/B	G#7add..	Gm7/C	A-7add..	A-m7	B7add#9	B-m7	<input checked="" type="checkbox"/> ['vii']	
C#7add..	C#m7/B-	Dm7/A	D-7add..	E7add#..	E/o7/D	Fm7/A-	F7/C	G#7add..	Gm7/D	A-7add..	A-/o7	B7addb9	B-m7	<input checked="" type="checkbox"/> ['vii']	
C7addb..	C#m7/E-	Dm7/C	D-7add..	E-7/B-a..	E/o7/G	Fm7/B-	F7/D	G7add#..	Gm7/E	A-7alte..	A/o7/E-	B-7add..	B-m7	<input checked="" type="checkbox"/> ['vii']	
C-7add..	C#m7/F	Dm7/E	D#7add..	E-7add..	E#m7	Fm7/E-	F7/E-	G-7add..	Gm7/F	A-7add..	A/o7/G	B-7add..	B/o	<input checked="" type="checkbox"/> ['vii']	
C7/B-	C#m7/F#	Dm7/F	D7add#..	E7addb..	E#m7	Fm7	F7/F#	G7addb..	Gm7alt..	A#7add..	A#m7	B-7add..	B/o7	<input checked="" type="checkbox"/> ['vii']	
C7/D	C#m7/G	Dm7/G	D-7add..	E-7add..	E#m7/B..	Fm7/D	F7/Galt..	G7/A	Gm7	A7add#..	A#m7	B7add#..	B/o7	<input checked="" type="checkbox"/> ['vii']	
C7/E	C#m7	Dm7	D#7add..	E7	Em7	F-m9	F7alter..	G7/B	G-m9	A-7add..	A#m7/..	B-7/Ead..	B/o7	<input checked="" type="checkbox"/> ['vii']	
C7/Ealt..	C#m7/..	Dm7/..	D7addb..	E7/A	Em7/A	Fm9	F-7	G7/Balt..	G-m9	A7addb..	A#m7/..	B-7add..	B/o7	<input checked="" type="checkbox"/> ['vii']	
C7/F#	C-m9alt..	D#m9	D7/A	E7/A#	Em7/B	Fm9	F-7alter..	G7/C	G-m9alt..	A7	Am7/C	B7addb..	B/o7	<input checked="" type="checkbox"/> ['vii']	
C7/F#al..	C#m9	Dm9	D7/E	E7/B	Em7/D	Fm9/B-	F-maj7	G7/C#	G#m9	A7/B	Am7/D	B--7add..	B#n	<input checked="" type="checkbox"/> ['vii']	
C7/G	C#m9	Dm9/A	D7/F#	E7/Balt..	Em7/G	Fm9/C	F#7	G7/D	Gm9	A7/C#	Am7/E	B7	Bm	<input checked="" type="checkbox"/> ['vii']	
C7+	C#m9/F	Dm9/C	D7/G#	E7/D	Em7alt..	Fm9/E-	F#7/A#	G7/E	Gm9/C	A7/D	Am7/F	B7/C	Bm7	<input checked="" type="checkbox"/> ['vii']	
C7alter..	C#m9/G	Dm9/G	D7/G#a..	E7/F	EmM7	Fm9/D	F#7/B#..	G7/F	Gm9/D	A7/D#	Am7/F#	B7/D#	Bm7	<input checked="" type="checkbox"/> ['vii']	
C-7	C#m11	Dm9/G	D7alter..	E7/G#	E-o7	F-m11	F#7/C#	G7+	Gm9/F	A7/D#a..	Am7/G	B7/E#	Bm7	<input checked="" type="checkbox"/> ['vii']	
C-7alter..	Cm11	D-m11	D-7	E7/G#a..	E-m9	F#m11	F#7/E	G7alter..	G-m11	A7/E	AmM7	B7/F#	Bm7	<input checked="" type="checkbox"/> ['vii']	
C-maj7	Cm13	D#m11	D-7/A-	E7alter..	E-m9al..	Fm11	F7/G	G-7	Gm11	A7/Ealt..	AmM7/C	B7+	B-n	<input checked="" type="checkbox"/> ['vii']	
C-maj7/..	C-dim	Dm11	D-7	E-7	E-m9	Fm11/B-	F#m7	G-7alte..	Gm11/C	A7/G	AmM7/..	B7alter..	B-n	<input checked="" type="checkbox"/> ['vii']	
C#7	C#dim	Dm11/F	D-7/A-	E-7/A	Em9	F-dim	F#m7	G-7	Gm13	A7+	A-m9	B-7	B-m5	<input checked="" type="checkbox"/> ['vii']	
C#7/B	C#dim/..	Dm11/G	D-maj7	E-7/B-	Em9	F#dim	F-maj7	G-maj7	G-dim	A7alter..	A-m9	B-7/A-	Bm	<input checked="" type="checkbox"/> ['vii']	
C#7/E#	Cdim	Dm13	D-maj7..	E-7/D-	Em9alt..	F#dim	F-maj7/A	G-maj7	G-dim/C	A-7	A#m9	B-7/E	B-m	<input checked="" type="checkbox"/> ['vii']	
C#7/F#	Cdim/F#	D-dim	D-maj7	E-7/E	E-m11	F#dim/..	F-maj7/..	G-maj7/C	G-dim	A-7/C	Am9	B-7/F	B-m1	<input checked="" type="checkbox"/> ['vii']	
C#7/G#	Cdim/G-	D#dim	D-maj7/..	E-7/F	Em11	F#dim/..	F-maj7/..	G-maj7/..	G-dim/..	A-7/D	Am9/D	B--7	B-m1	<input checked="" type="checkbox"/> ['vii']	

In this interactive interface, the user can achieve similar information for chord alternatives. Here the chords are organized by interval.

Future Considerations

Immediate improvements to this project should be directed towards the weight management of perceptrons surrounding rare chords in the dataset. There are multiple options to handle this class imbalance. One solution involves upsampling the data that has rare chord occurrences. This can be done at the song level or the N-length sequence level as long as the y prediction is also multiplied. Another option is to handle the weights within the neural network by using Sklearn's `compute_class_weight` function. This function takes parameters that will normalize weights by default or follow a predefined dictionary of classes and weights.

In terms of data management and cleaning, more songs can always be used and even organized to train the network using different genres of music. Chord definitions could be further cleaned to consolidate more chords based on similar features.

Further expansion of features would include a web interface that combines all forms of chord classification and prediction into one location. The added option to transpose into any other key would also be widely useful for most musicians. A wider chord vocabulary in the key of C is not as useful for any musicians aiming to play songs in any of the other 23 keys.

Conclusion

RNNs provide a very useful platform to make music appreciation more accessible across all aspects, whether it be exploring new sounds, or learning music theory from an entirely new angle. It is also exciting to be able to explore what progression features might influence our musical preferences. Perhaps one enjoys most songs that utilize the 9th interval added to chords. We can then explore musical genres that use the 9th in different applications.

There is still plenty to explore in the realm of visualizing music and discovering what new genres could emerge by the use of rare progressions.