Music Analysis and Generation

Karo Castro-Wunsch

Objectives

Given the somewhat exploratory nature of this project, it's goals have been realigned and updated over the course of the timeline. The primary goals visited are:

1. Given a database of music, generate and mix a playlist of songs.
   1.a. Detect appropriate points to transition from one song to the next.
   1.b. Choose and order songs to build a setlist using song features
2. Implement an LSTM using TensorFlow and develop a data pipeline for it.
3. Given a database of music, generate a novel piece of music in a similiiar style.
4. Transfer the style of one piece of music to another.

1 Phrase Detection using Similarity Matrices

Previous Work

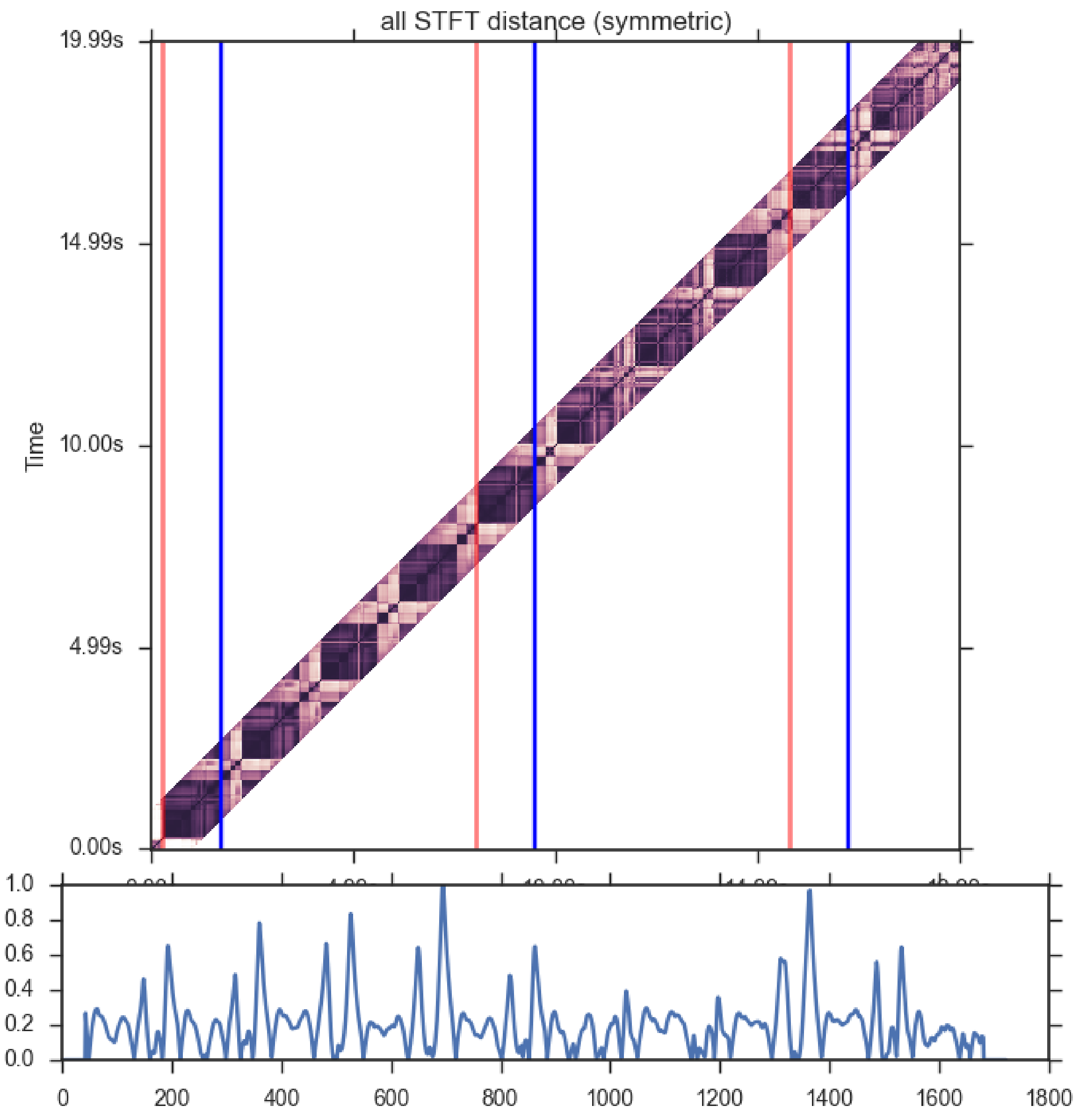
Previous attempts at music segmentation include an approach using the detection of breaths (the inhale or exhale of an artist) to break up a peice [1] (https://www.researchgate.net/publication/288593236_Automatic_phrase_segmentation_of_MP3_songs_based_on_the_technique_of_breath_sound_detection), and a larger group of research that focuses on using Similiarity Matrices (S-Matrices) to perform segmentation 2. This S-Matrix approach is what the phrase detection in this project is based off of.

1.1 Build S-Matrix

The S-Matrix is Constructed by first taking a Short Time Fast Fourier Transform (STFT) of the waveform of a song and then computing the cosine similarities of each of the resulting Fourier Coefficient vectors with respect to each other. For completeness sake, the STFT is the Fast Fourier Transforms of the segments of a waveform split into k-length segments. The cosine similarity of two vectors is computed using scipy's

```
spatial.distance.cosine
```

The S-Matrix is then a matrix with both x and y axes representing the STFT and cells of the matrix representing the scalar distance between its respective x and y STFT vectors. This means the matrix is symmetrical about x = y.



In the computation of the S-Matrix, the algorithm was improved for runtime by noting that the S-Matrix's symmetricality means we only need to compute half the matrix. Also, since we will only use a diagonal slice of the S-Matrix in order to compute the Novelty Vector, we only need to compute similarities for cells along this line. This reduces the complexity from O(n^2) to O(n). In the above figure, the red lines indicate the actual phrase transition points, while the blue lines are the algorithmically determined transition points. This is a case where the algorithm determined *almost* ideal results with the appropriate intervals with a slight offset, this results in a very low score, despite the relatively good quality of the results. This sort of error could be compared with a person clapping to the offbeat of a song.

1.2 Build Novelty Vector

In order to utilize the S-Matrix, we must detect where the similarity between segments of STFT slices changes abruptly. The vector of the correlation each segment of the S-Matrix to the next segment will be called the Novelty Vector. To compute this, we compute the cosine similarity between a checkerboard matrix (insert pic of checkerboard matrix) and the sequence of matrices centered along the diagonal line through the S-Matrix x = y. This gives us a vector with peaks at the points where segments change. The size of the Checkerboard Matrix used determines the scale of the repeating features detected. A c=0.1s matrix detects individual notes, where a c=3s matrix detects phrases. A matrix of size 3s was chosen. The matrix size was determined by hand tuning but a better result could likely be achieved by learning the size. Of note is that the first and last c/2 seconds of the Novelty can not be computed, so a smaller checkerboard matrix size is better as well as faster. In the above figure, the novelty vector is the line plot beneath the S-Matrix.

1.3 Choose Novelty Peaks

The S-Matrix and Novelty construction were fairly reliable, albeit slow, however choosing the peaks of the Novelty Vector is a more subjective task because it relies on some prior knowledge of music segmentation. Because there are frequently 'false' peaks in the Novelty, music can not simply be segmented at each peak. Heuristics were then integrated into peak picking in order to get better results. Peaks satisfying the following conditions were favoured:

- Peaks falling on beat (beat detection was used)
- Peaks spaced in even intervals
- Peaks spaced in 4, 8, 12, or 16 bar intervals (conforming to western music)

Clustering was also implemented in order to convert groups of weaker peaks into a single more relevant peak at the position of the cluster mean. The chosen peaks were then determined to be the beginning/end points of the musical phrases of the input song.

1.4 Validation

In order to validate the results of algorithm, a batch of 10 songs were hand labelled. The algorithm had an average error of 0.61, meaning 61% of peaks were incorrectly chosen. This error was difficult to calculate because certain errors, such as the each of the peaks being off by the same interval but the overall structure being correct resulted in a poor score even though the algorithm actually performed relatively well. On the other side of things, in cases where the algorithm chose too *many* peaks, it had a good chance of choosing the correct peaks but also including many unnecessary peaks. This would result in an artificially high score. Additionally, songs with vocal segments are particularly hard to process because human vocals add additional complexity to the sound structure beyond that of most instruments. Using the algorithm exclusively on electronic music with no vocals significantly improved its performance.

1.5 Song Transitions

Using the above algorithm to detect the phrases in two songs, we can line up the songs in order to transition from one to the other in a smooth fashion by cutting each the songs off at their phrase transition points and adjusting the tempo of the songs so that they match. In the accompanying music clips, the hard-cut functionality is demonstrated. To reduce computation time, here only the first and last 30s of each piece are analyzed. The two songs are tempo matched to a bpm close to both songs. Once both songs are analyzed for their phrases, the first and last phrase transitions, respectively, are chosen to be the hard-cut points. Since peaks are only chosen if they are distinct (the peaks have a reasonably high confidence of importance), the first and last peaks are start/end points. This means that intros and outros are cut-off and not grouped into the rhythmic body of the song. The hard_cut_0 does an effective job of cutting at phrase transition points smoothly and is non-jarring. hard_cut_1 seems to cut off awkwardly as it chooses an end point of the first song after the song's beat end, on a trailing note. hard_cut_2 displays a limitation of this technique in that it chooses a musically logical place to cut which falls in the middle of a sung word. There is no way for the algorithm to identify speech patterns.

2 Sequence LSTM

In order to come to a better understanding of Neural Network Architectures and the TensorFlow library, I implemented a variety of different NNs starting with very simple and culminating in an LSTM that I paired with various data sets. The sequence to sequence LSTM model implemented is based off of char-rnn-tensorflow and TensorFlow-Examples.

2.1 Sequence to Sequence

The sequence to sequence LSTM model implemented is based off of char-rnn-tensorflow and TensorFlow-Examples. In order to understand the model, Sine Wave data was generated in order to test its sequence predicting capabilities. This was tested in both discrete and gradient environments. Additionally, the model supports both sequence predicting and sequence classification functionality. In order to do this, different final layer activations are implemented. For classification, a softmax is used, whereas for sequence prediction a sigmoid is used, which better handles outputs that do not represent a probability distribution. Testing the model first on datasets of smaller dimensionalities than music (waveform STFT vectors can be reduced to ~500 points at the smallest) allowed me develop the model faster and detect impending difficulties, the most significant of which is the limited nature of the LSTM's long term dependency detection capabilities (despite it's 'long term' memory).

3 Music Generation

Inital testing was done by working off the GRUV project which explores music generation using raw waveforms as data. This project uses a single layer LSTM implemented in theano with Keras. Experimentation suggests the network has the capacity to memorize simple sequences of music (simple drum tracks) with a relatively low level of noise. The more complex and less repetitive the music gets, the more noise is introduced to the model. The model used for testing had 1024 hidden units per layer extending 40 timesteps. Given blocks of 2048 samples and a sample rate of 44100 this gives the network a 1.85 second memory, this is reflected by the inability of the network to capture musical features longer than a few notes.

4 Style Transfer

This is the ongoing task of the project, following the methodology outlined in A Neural Algorithm of

Artistic Style, transferring the technique from the image domain to the audio domain.