Melody Extraction from Generic Audio Clips

Thaminda Edirisooriya, Hansohl Kim, Connie Zeng

Introduction

In this project we were interested in extracting the melody from generic audio files. Due to the context-dependent nature of music, particularly the melodic voice, we decided to use a recurrent neural network (RNN), which keeps a memory of previous inputs. The input layer to the network captures a time step of a song, containing the distribution of frequencies, the main chord, and the single loudest frequency in that time step, which are all preprocessed from raw audio. The output layer is a softmax classification of the note predicted to be the melody. We implemented this in Python, with several modules from NumPy [1] and SciPy [2].

Related Work

One of the most accurate methods to date, from Salamon, et al. [3], computes the harmonic sum of each pitch as a salience function to estimate its likelihood in an audio signal. The most salient pitches are then grouped into continuous contours, and the contours are filtered based on characteristics such as average pitch and pitch deviation. This is well suited for vocal melodies, but performs poorly on instrumental music, which has more overlapping voices and sudden changes.

Bosch and Gómez [4] created a variation of this method, designed to improve performance on an orchestral dataset. Instead of using harmonic summing, the audio signal is modeled as a lead plus accompaniment, with the lead further approximated as a source-filter model. The pitch salience is then calculated using maximum likelihood. As expected, this method yielded better results on orchestral music, while still maintaining moderate accuracy on vocal music.

Other approaches include that of Arora and Behera [5], which finds harmonic clusters and thresholds them based on their summed squared amplitudes, and that of Tachibana et al. [6], which separates out sustained harmonies by using varying time windows, and then separates out aperiodic percussion by using varying frequency windows. These methods are more mathematically straightforward but rely on simplified assumptions.

Our work is similar to that of Salamon's, but instead of manually specifying the salience function, we want to see if an RNN can learn to identify the notes from an audio sample. There are many interactions between sound frequencies that may be hard to model mathematically, so a neural network might be able to perform better on this task. Johnson [7] has done related work, using a bidirectional RNN to compose music.

Dataset

Our primary dataset is a collection of Bach chorale harmonizations, from `jsbchorales.net` [8]. These chorales have harmonic patterns that are still widely used and can help make inferences about the melody. They are in MIDI form, with one track per piece containing the melody. We converted each raw MIDI file into a WAV file as a sum of sawtooth waves, and also created a WAV file containing the isolated melody as simple sine waves.

We supplemented our dataset with clips used in the annual Music Information Retrieval Evaluation eXchange (MIREX) melody extraction task [9]. They are drawn from a variety of genres, including pop, jazz, and classical. These already come as WAV audio files, along with a reference file containing the annotated melody.

Since melody extraction is a complex task, we started by focusing on the chorale melodies. We used a training set of 80 chorales, from which 10% was randomly selected for the validation set, and a test set of 20 chorales. Then to start learning more general melodies, we added 12 MIREX clips to the training set and 3 MIREX clips to the test set.

Feature Extraction

We took the fast Fourier transform (FFT) of the full audio WAV files to obtain a matrix of all constituent frequencies over 0.25-second time steps. For the chorales, we similarly took the transform of the melody WAV files to identify the main frequency of the melody at each time step. For the MIREX clips, we simply read the frequency values of the melody from the reference file. As a simplification, we only considered frequencies from 0-2000 Hz, based on the typical range of music.

The full audio FFT was fed to a chord-recognition SVM to identify the main chord per time step. Each input vector to the neural network then consisted of the FFT (2000 features), the predicted chord from the SVM (24 features), and the frequency with the highest amplitude (1 feature) for a given time step. All time steps of a song were combined into a single matrix and stored with the target melody frequencies in a CSV file.

Chord-Recognition SVM

As part of our feature extraction, we created an SVM to predict the most likely chord in an audio sample. The SVM we used operates by solving the optimization problem:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1,\ldots,m \\ & \xi_i \geq 0, \quad i = 1,\ldots,m. \end{aligned}$$

Which is can be expressed as the dual optimization problem:

Implemented using the scikit-learn SVC class [10], the SVM takes an FFT as input, and outputs an indicator vector with one of 24 chords chosen. The 24 possible chords come from 12 half-step base notes and 2 tonalities, major or minor.

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1,\ldots,m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0, \end{aligned}$$

To generate data for the SVM, we created chords by summing sawtooth waves and taking the FFT. For each base note and tonality, we considered the three notes that make up the corresponding chord and, at each octave, included each note with a random chance. We used a span of 6 octaves, from 3 below middle C to 2 above middle C, as a reasonable range for chords in actual music. This process essentially creates a random sampling of all the permutations and multiplicities of a chord.

Kernel	Accuracy
RBF	80%
Linear	83%
Polynomial	51%

Table 1: Comparison of kernels for the chord-recognition SVM.

We used a training set of 200 samples per chord and a validation set of 50 samples per chord to select the kernel for the SVM. As shown in Table 1, the linear kernel resulted in the best classification. We further tested it on a set of more complete chords, containing at least one instance of the base note and the middle note. The SVM achieved 97% accuracy on a set of 50 samples per chord. Finally, we tested a small sample of Bach chorales, with 86% accuracy.

Percussion Processing

As part of the preprocessing on pop audio samples in particular, we tried to remove the spikes in frequency that occur when loud percussion such as a kick or snare drum occurs. A kick or snare often has high power levels in the frequency ranges also shared by vocal and other instruments, and can thus obscure the melody information that we want to extract. This is a problem because the chorales did not contain percussion, and thus our algorithm did not generalize too well to percussion heavy music.

In an attempt to correct this issue, we created a model of the FFT of a kick drum and a snare drum by taking the means of the FFTs of 500 different kick and snare samples respectively. We used this to detect percussion in audio samples by taking the FFT of 0.05-second time steps of the audio, and computing the dot product with the kick or snare model as a similarity measure. The higher the value, the more likely it was to contain a kick or snare sample. We then chose the median of every 5 time slices (in terms of similarity) as a representative of the entire 0.25-second time step, to avoid spikes due to loud percussion.

However, when we tested this method, we found no improvement on our accuracy. We noticed that the heuristic we used to remove percussion also removed important melody information. Future work could seek to refine this process as a means of improving the generalization of the algorithm as a whole.

Recurrent Neural Network

The 2025 features from preprocessing are input into our RNN (implemented with PyBrain [11]) with a 17-node hidden layer and an output layer of 60 classification nodes (see Figure 1). The hidden layer is split into two sections based on connectivity. The 5-node "octave layer" only connects to the 2000 frequency inputs, and each "octave" node only outputs to 12 of the 60 output notes, corresponding to an octave. The 12-node "note" layer is fully connected to the input and output. When activated, the output node with the maximum value is taken as the predicted melody note.

Hidden layer nodes are implemented as Long Short-Term Memory (LSTM) recurrent modules. These miniature networks can store input values for near-arbitrary lengths of time. LSTM architectures vary, but all include "gate" nodes

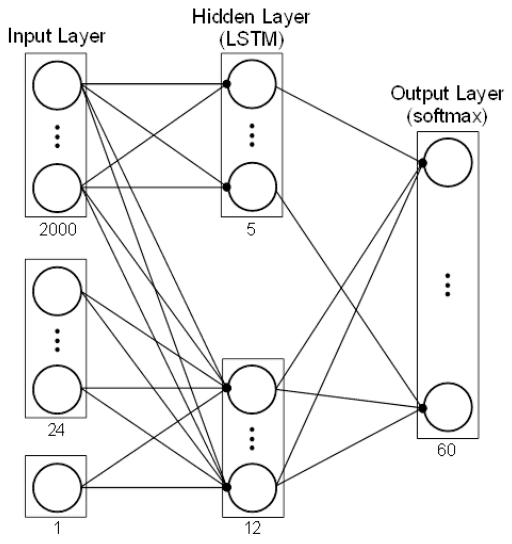

Figure 1: RNN Structure

that control when an input will be remembered, when an input should be forgotten, and when the memorized value should be output. The frequency information in music can fluctuate wildly, so these LSTM recurrent modules provide the potential to judge when a change in inputs should be ignored and when a change is significant enough to warrant changing the note of the predicted melody.

The 60 output nodes represent 60 possible output notes, and they use the softmax equation:

$$\text{softmax}(x) = \frac{\exp(-\mathbf{w}^T x_j)}{\sum_k \exp(-\mathbf{w}^T x_k)}$$

This is a multi-class generalization of the sigmoid classifier, and all nodes sum to 1, representing the estimated probabilities of each class. The maximum value output node is taken as the output note.

The neural network is trained with Resilient Backpropagation, which is variant of the standard backpropagation algorithm. The weights are updated by multiplying by one of two constants decided by the relative signs of the gradient in the current and previous iterations, while the magnitude of the gradient is ignored. Resilient Backpropagation was chosen for its speed and its ability to store individual learning rates that can adapt for each weight.

Results

We trained the neural network for 525 iterations, using a validation set to estimate convergence. The training curve, in Figure 2, shows that convergence was reached near 100 iterations.

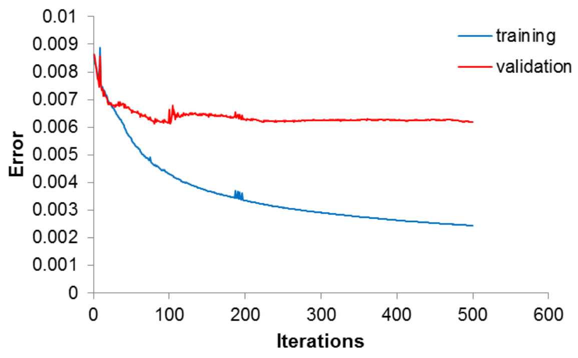

Figure 2: Training Curve

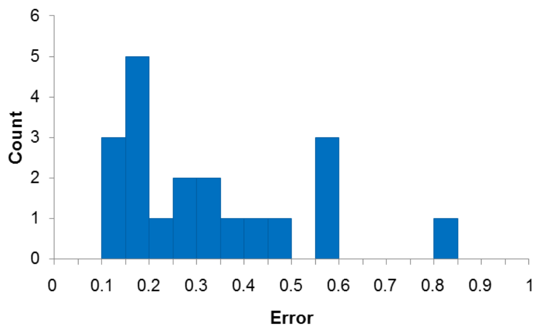

Figure 3: Error Rate Histogram

The average classification error on the test set was 33%. This relatively low error rate likely demonstrates that the neural network has found and taken advantage of patterns in the frequency and chord data to aid in predicting the melodic pitch. Looking beyond the average of the error rate, we note that 40% of the test chorales are clustered below 20% classification error, with four outliers accounting for much of the error. This indicates that the trained net is accurate on many of the chorales, but also fails to grasp certain anomalous patterns.

Note that the classification error rate alone does not take into account similarities between certain notes. The note C is more different from a D than it is from a C one octave above. Examining a sample predicted melody, we note that the prediction is generally accurate, including the absence of melody at the start, but a closer look at some of the errors reveals a pattern to some of the missed predictions.

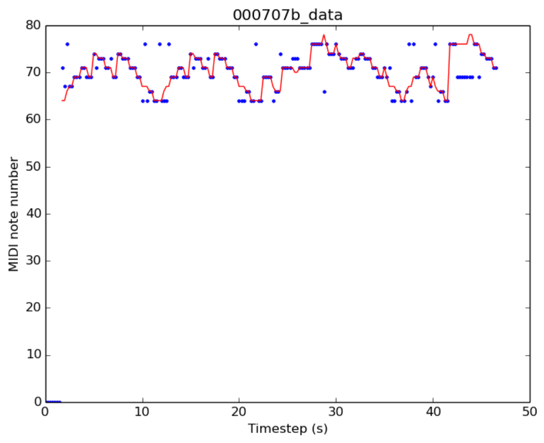

Figure 4: Predicted Chorale Melody

Here, the note 64 in the melody is often (and in a regular repeating pattern) wrongly classified as note 76. This is in fact the same note, transposed one octave up. This provides further indication that our trained network is learning harmonic patterns. Examination of the wrongly classified notes often reveals a shift of 4, 5, or 8 half steps, which correspond to the minor and major third as well as the dominant fifth, all of which play important roles in established harmonic rules.

The results up to this point have been from testing on chorales. One of our motivations was to use the chorales and harmonic training as a base for later extension to other genres and styles of music. While we do not expect the classification to perform as well on other styles of music, we are interested in testing how useful the harmonic patterns in chorales are for melody extraction. Below is a predicted melody for a MIREX track, from an RNN trained on the dataset containing 12 MIREX clips:

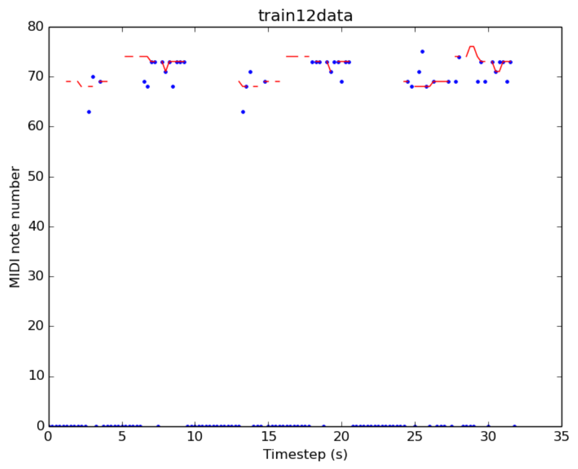

Figure 5: Predicted MIREX Melody

From the results we see that the neural network is capable of generalizing some of its predictive power to music other than chorales. The predictions are not only able to predict the predominant tones in the melody, but also capture several nuances in the melody. This is an encouraging result in terms of future work, and may be coupled with other methods such as a more developed percussion processor in the future.

Conclusions and Future Work

From our results, we conclude that the use of harmonic data in addition to raw frequency data, fed through a recurrent neural network, can be effective at predicting the melody of harmony-rich pieces. The Bach chorales represent a relatively clean audio track, without the added complication of atonal elements such as percussion. In addition, the Bach chorales adhere to established harmonic rules strongly, but this is also true of much of modern popular music.

The harmonic patterns learned and recognized by our neural network, and the substantial preprocessing system used to extract valuable features, represent a foundation for future work in generalizing the system to a wider range of music. As seen with the test on MIREX samples, the harmony-based system can be effective beyond the chorales. Improvements to the percussion processing system or more powerful chord classifiers that account for chord beyond simple majors and minors are possible future works that could extend the effectiveness of this system to a more general body of music.

References

[1] Stéfan van der Walt, S. Chris Colbert and Gal Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation." *Computing in Science & Engineering* 13 (2011): 22-30

[2] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2015-12-10].

[3] Salamon, Justin, et al. "Melody extraction from polyphonic music signals: Approaches, applications, and challenges." *Signal Processing Magazine, IEEE* 31.2 (2014): 118-134.

[4] Bosch, J., and Emilia Gmez. "Melody extraction by means of a source-filter model and pitch contour characterization (mirex 2015)." *Music Inform. Retrieval Evaluation eXchange (MIREX)* (2015).

[5] Arora, Vipul, and Laxmidhar Behera. "Online melody extraction: Mirex 2012." *Extended abstract submission to the Music Information Retrieval Evaluation eXchange (MIREX)* (2012).

[6] Tachibana, Hideyuki, et al. "Melody line estimation in homophonic music audio signals based on temporal-variability of melodic source." *Acoustics speech and signal processing (icassp), 2010 ieee international conference on.* IEEE, 2010.

[7] "Composing Music With Recurrent Neural Networks." Hexahedria. 03 Aug. 2015. [Online; accessed 2015-11-09]

[8] https://web.archive.org/web/20110930/ http://www.jsbchorales.net/sets.shtml [Online; accessed 2015-11-20]

[9] http://labrosa.ee.columbia.edu/projects/melody/ [Online; accessed 2015-11-20]

[10] Pedregosa et al., "Scikit-learn: Machine Learning in Python." *JMLR* 12 (2011): 2825-2830.

[11] Schaul et al., "PyBrain." *JMLR* 11 (2010): 743-746.