**1. LSTM**

I have found some interesting papers about LSTM. Some of them provide the Python code, pseudo-code or back-propagation algorithm for LSTM [5-8]. I tried the codes or wrote codes according to the algorithms in [5-7], but became frustrated.

Finally I found the reason. On page 24 of paper [7] it is stated that RNN (and LSTM) has 5 categories, where most of these codes realize the classification problem (b) which is multi-input, single output. What we want to solve, however, is the generative model problem (e), which is multi-input, multi-output, and the output is instant. This is the reason that most LSTM codes do not fit what we want.

However, I have got an RNN code which solves problem (e) quite well. To test the music model, we can simply use RNN first, because when we want to verify the feasibility of a new model, using either RNN, LSTM or URNN will not be critical on the performance.

**2. Music Model: Training**

My music model has 3 tracks: Rhythm, Chord and Melody.

First let's talk about training, which does not need much music knowledge and special adjustments. As usual, the input and output vectors are inticator vectors, which is all zero but a 1 in a certain element.

2.1 Rhythm

Rhythm has input/output vector length of . Each vector represents a bar, and each element represents the rhythm pattern of the bar. We assume that melody notes' values are multiple of 8th notes, therefore each bar (a full note length) has 8 available time points for melody notes. The index of the non-zero element can be written in a 8-bit binary code, and its digits of 1 show that there is a melody note at this time point.

2.2 Chord

Chord has input/output vector length of . Each vector represents a bar too. We assume that chord notes' values are multiple of 2nd notes, therefore each bar has 2 available time points for chord notes. Chord notes are considered to be always present, and if a chord is longer than 2nd note, it is separated into 2nd notes of the same chord.

Let's only consider 8 most common chords: C, G, F, Am, Em, Dm, E, A. Dominant 7 is regarded as Major chord (G7=G). In East Asian pop music we can hardly find anything else, unless there is a musical modulation.

Back to the vector, the index of non-zero element can be written in a 2-bit octal code, and its each digit (0,...,7) indicates the chord at this time point. If there is an abnormal chord in this bar, it is represented as the last number 65(decimal), and will neither be played nor give preference on melody.

2.3 Melody

Melody has input/output vector length of . In melody, each vector represents a note. Currently I only consider the notes within the range from great C (130.81Hz, midi number 48, one octave below central C) to two-lined c'' (1046.5Hz, midi number 84, two octaves above central C). If I find notes outside this range when I am making the database, I will expand it accordingly. The vector representation is quite intuitive.

**3. Music Model: Generating**

The generation is similar to the RNN-based generative language models. However, I will introduce some terms (adjustments): Harmony, Preference, Copy, and Hard Limit (the names are temporary).

The generating order is: Rhythm -> Chord -> Melody -> wav file. The reason that we generate melody at last, is that the length of melody notes is determined by rhythm, and there are other properties affected by rhythm and chord.

3.1 Harmony

The harmony adjustment is decided by chord, and only applies on melody. After we generated rhythm and chord, we can calculate that for each melody note, which chord is active. The pitches that are equal to or octaves higher than pitches of chord notes are harmonic, and their log-likelihoods are increased.

3.2 Preference (planned for next week)

I would like to introduce pitch preference and rhythm preference.

Pitch preference adjustment applies on melody. For a 40-bar pop song, the common pattern is a 16-bar verse, 8-bar prechorus, and a 16-bar refrain. I would like to assume:

|  |  |  |  |
| --- | --- | --- | --- |
| Part of song | Preferred pitch | Midi number | Frequency |
| Verse | G --- g | 55 --- 67 | 196 --- 392 |
| Prechorus | c --- c' | 60 --- 72 | 261.63 --- 523.25 |
| Refrain | g --- g' | 67 --- 79 | 392 --- 784 |

since that from verse to prechorus to refrain, the song goes more and more exciting.

The preferred pitches have their log-likelihoods increased.

Rhythm preference applies on rhythm, and will be introduced later.

3.3 Copy adjustment (planned for next week)

The copy adjustment applies on rhythm and chord, and it may or may not apply on melody (it will be decided by my experiment result). Copy adjustment says that, it is assumed that each bar is likely to appear again after 8 bars, if it is still in the same section (verse or refrain). The elements appeared 8 bars ago will have their log-likelihood increased, the detail will be discussed later.

For songs of variable length, I assume that:

If the verse is at least 16 bars, the copy is in a unit of 8 bars. If once there are less than 8 bars left, they will not copy.

The bars in prechorus never copy.

If the verse is longer than 8 bars, the copy is in a unit of 8 bars. If once there are less than 8 bars left, they will copy the end of the previous 8-bar unit.

In the example of 40-bar song, it becomes: the 9th-16th bars tend to copy the 1st-8th bars; the 33rd-40th bars tend to copy the 25th-32nd bars.

3.4 Hard limit (partly planned for next week)

Hard limit is intuitive. It forces certain entries to be fixed.

Now only a few elements have hard limits.

The first bar, the first bar of refrain, and the last bar are forced to have all Major C as chords.

The last bar is forced to have [10000000] as rhythm, which means there is only one full note at the beginning of the bar.

The last note is forced to be one-lined c' (midi number 72, 1 octave higher than central C).

3.5 Details

Here are some detailed discussions about picking rhythm and chords.

We have 2 possible methods to do that. One is to directly pick one according to the probability from the RNN output. The other is to calculate the probability for every state of each possible note, and then sample from this distribution.

I prefer the second method. The first reason is that, it can give us more diversity, especially on rhythm, when there are some very different patterns both with high probabilities.

The other reason is that, it can make things easier when we make adjustment to the likelihood. We just operate on single element and it is intuitive. We do not need to calculate and map it to the vector.

Also, for rhythm, the activity of each 8th note is sampled from a Bernoulli distribution. We can just sample from a Uniform distribution on [0,1] and compare the random number with the Bernoulli probability. It is very simple to make the toss unfair (by calculating the 1.1-th or 0.9-th power of the uniformly random number), to show our preference on the rhythm, by making it more or less likely to be active.

3.6 Making wav file

I use Matlab to do this. I use 16KHz sampling rate and 120bpm rhythm (which can be changed). Each note of the triad chord has 1/3 the amplitude of the melody. The chord is remapped onto the octave of Contra F to Great E (midi number 40 to 52, frequency 87.31Hz to 164.81Hz).

In the attachment is the song generated (from only one song as training dataset).

References:

[5] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures[J]. Neural Networks, 2005, 18(5): 602-610.

[6] Lyu Q, Zhu J. Revisit long short-term memory: An optimization perspective[C] //Advances in neural information processing systems workshop on deep Learning and representation Learning. 2014.

[7] Lipton Z C, Berkowitz J, Elkan C. A critical review of recurrent neural networks for sequence learning[J]. arXiv preprint arXiv:1506.00019, 2015.

[8] Gers F A, Schmidhuber J, Cummins F. Learning to forget: Continual prediction with LSTM[J]. Neural computation, 2000, 12(10): 2451-2471.