CS230

Automatic Chord Arrangement from Melodies

Yulou Zhou, Shuxin Meng
Department of Computer Science
Stanford University
{ylzhou, shuxin}@stanford.edu

Abstract

We developed an automatic chord arrangement tool that uses a deep neural network learning to predict chord for a given melody. Our final model is a sequence classification model consisting of two long short term memory (LSTM) encoders, one for previous chords, one for melodies from the current measure. It is trained on a music dataset from Wikifonia. Our project achieved a point-wise matching accuracy of 33.6% and a harmonious estimation of 60.5%. Considering music composition as a creative process, these results can be interpreted as meaningful. [1]

1 Introduction

Imagine that you have a beautiful melody in your mind. You want to arrange accompaniment and eventually make it as a finished song, but if you lack training in music, this task would become a mission impossible. Compare to writing melodies, arranging chords for composed melodies often causes more difficulties for music learners, due to their lack of music theory knowledge. In this paper, we will address this challenge with an automatic chord arrangement tool that uses a deep neural network to generate chord for a given input melody. Our purpose is to eventually design a tool that can both give new learners suggestions of chord arrangement and provide skilled composers with inspirations.

Arranging chords for a melody involves both creativity and rules, which makes the project more challenging. On the one hand, generated chord progressions have to follow many harmonic and progression rules. On the other hand, there are no unique correct answers for chord progressions, so evaluating the performance of our model with a variety of "good answers" is challenging.

2 Related work

In 2011, Chuan et al. proposed a hybrid system for generating style-specific accompaniment, which can learn a musical style from several examples and then create accompaniment for melodies. [1] The system first finds chord notes and uses neo-Riemannian transforms between checkpoints to generate possible chord progressions, and then selects the final chord progressions by the Markov chain.

Lim et al. introduced a novel approach for generating a chord sequence from symbolic melody using BLSTM model. [3] The result shows that BLSTM achieves the best performance followed by DNN-HMM and HMM. They concluded that the recurrent layer of Bidirectional LSTM is more

[1] GitHub Repository: https://github.com/luggroo/230-melody-to-chords

appropriate to model the relationship between melody and chord than HMM-based sequential methods.

Yang et al. proposed to using paired model and multi-task learning for melody-to-chord in their cs224N project at Stanford. [2] The paired model takes a pair of melody and chord lines as input, and output how good a match the incoming melody-chord pair constitute and additionally how to modify the inputs chord to get a better match.

3 Dataset and Features

We found nearly 7,000 lead sheets of popular music data in musicXML format provided by Wikifonia, a public music sheet database. It terminated in 2013, but fortunately, the database was preserved. The files contain melodies labeled with chords. But we soon realized that the dataset includes a mix of unbalanced genres, including pop, jazz, and folk songs. These different genres will result in a different distribution of chords. Besides that, since Wikifonia was a platform that everyone can upload their music, the problems of poor quality and redundancy is very prevalent. Fortunately, we found that the Music Audio Research Group in Seoul National University posts the partly processed and selected data from Wikifonia. This dataset, which we are using, consists of 2,252 lead sheets all in major keys, and the majority of the bars in the lead sheets have a single chord per bar. We split the data into two sets – a training set of 1802 songs or 72,418 bars and a test set of 450 songs or 17,768 bars.

We extracted time signatures, melodies, keys, and chords (types and root notes) from the dataset. We converted all notes and rests to equal temperament numerical expressions (0-13, ignore octaves), divided the duration of notes by each rhythmic unit (time signature) as one-note entrance per unit. We also built a chord map for all the type of chord in our dataset and converted labeled chords to a list of root notes and intervals. For example, a major chord would be [root, +4, +7].

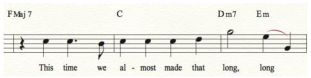


Figure 1: Original data

1	4/4	[0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 10]	[12, 4, 7]
2	4/4	[9, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]	[2, 3, 7]
3	4/4	[0, 0, 0, 0, 0, 0, 9, 9, 9, 9, 10]	[10, 4, 7]
4	4/4	[9, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7]	[5, 4, 7]
5	4/4	[9, 9, 9, 10, 10, 10, 9, 9, 9, 7, 7, 7]	[12, 4, 7]

Figure 2: Processed data

4 Initial Exploration: Seq2seq Tagging

4.1 Model Design

We implemented our baseline model based on a Pytorch sample model for part-of-speech tagging. The model takes a certain-length sequence (window) of notes as input, and assign pre-trained integer embeddings for each note. Then, an LSTM takes the matrix word embeddings as inputs and output hidden states. A following linear layer mapped each hidden state into a tag space vector. Finally, it uses a log-softmax function to determine the tag scores. The output is a one-hot matrix (input size * 13), which is then converted to a sequence of root note labels by an argmax function.

4.2 Experiments and Baseline Results

We ran our baseline models with very small embedded dimensions, and experimented on several sizes of hidden dimensions and learning rates. For training-set evaluation, we used the most strict metric: the point-wise similarity between the true and predicted chord labels. A window size of 64, a learning rate of 0.1, a hidden dimension of 30, and an embedded dimension of 4 altogether achieved an accuracy of 0.295. Since chord arrangement is a creative process, such a result can be interpreted as a working one (a random generator would have an accuracy of 1/13, or around 0.077).

Our baseline model assigns a chord label for each note. We realized that this method does not match the usually process of chord arrangement. Instead, musicians usually assign a chord for each measure. Thus, the input features and output format of our final model should better match the natural process of chord composition.

5 Final Model: Sequence Classification

We chose to change adopt a model of the sequence classification one for our final model. Since previous chord progressions and current notes are deciding features for chord arrangement, our sequence classification model uses a bar as the input unit, takes both features as input, and outputs an entire chord as the predicted chord label for the bar. We evaluated various potential ways to represent the input and output features and the arrange the architecture of the model.

5.1 Feature representations and model components

5.1.1 Previous chords

Previous chords are crucial to the selections of chords. We initially attempted to represent previous chords as an array of notes in integer formal (e.g. C Major would be [1, 5, 8]), and directly feed the array into the final linear layer along with the output from LSTM. This approach generated few meaningful outputs. We attributed this to the representation of a chord as three individual notes, which do not capture the relationship between the root note and other notes in the chord. Thus, we decided to represent a chord in the format of [root note, interval to the second component, interval to the third component] (e.g. C major [1, +4, +7]), which better captures the relation between root notes and other chord components.

We also decided to represent previous chords as a series of one-hot arrays. If we send our model two previous chords, both C Majors, the input would be [1, 4, 7, 1, 4, 7], but in one-hot format. Then, we fed the one-hot matrix into another LSTM, and concatenated the final states of the LSTM for previous chords ("prev-LSTM") and the LSTM for the melody ("melody-LSTM") as the input to the linear layer. This modification brought us some initial working of 0.20 accuracy after just 300 training loops.

5.1.2 Various-length melody input

Melody lengths per bar vary across songs. However, mini-batch gradient descent with the "dataloader" class of Pytorch requires uniformed length for all input. Thus, we padded the end of short melody inputs with zeros. Previous experience shows a large amount of sequence-final paddings would hurt the quality of LSTM output, so we designed two ways for melody-LSTM to process the padded melody input.

The first method is slicing the input to only preserve the first n (n is around 8) notes of each bar. This method is motivated by our experience that the first few notes in a bar are usually most crucial to chord assignment. The second method is using the Pytorch methods of "pack_padded_sequence" and "pad_packed_sequence," which essentially deletes paddings before sending the melody into melody-LSTM, and pad zeros to the output. This approach, the implementation of which was very time consuming, also seemed ideal since it preserves the complete information of the melody input.

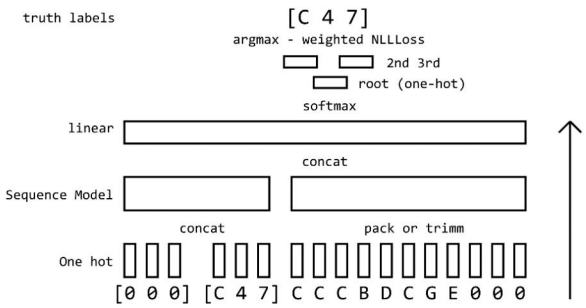


Figure 3: Sequence Classification Model

5.2 Summary of Model Architecture

Our model consists of two LSTM and a linear layer. First, a weighted "nn.Embedding" convert both the previous chord progression and bar melody to one-hot matrices. Then, two LSTMs (unidirectional and bidirectional are both tested), melody-LSTM and prev-LSTM, each takes in previous chord one-hot matrices and normalizes melody one-hot matrices. The final non-padded hidden-state of their output are concatenated together into a one-dimensional array, which enters the linear layer. The outputs of the linear layer go through a softmax activation layer to predict the root note. Then, we concatenated the root note to the outputs of the LSTMs, to generate predictions of the second and third notes through two other other linear layers.

6 Experiments and Results

6.1 Hyperparameters

We primarily used a batch size of 512 bars in the later stage of training. Smaller batch sizes brought severe oscillation to the weight decrease, and any batch size equal or greater than 1024 took too long to train.

Another important hyperparameter is the number of previous chords that feed into the model. Too few previous chords give too little context, yet too many previous chords cause the initial bars to receive less information than others and add burdens to computation. Testing shows that 6 previous bars is a good number.

Initially, our model tends to predict common root notes (such as C and G) and chord types (mainly major chords). Thus, we did statistical analysis on the proportions of notes and chord types and calculated the probability (p) of each root note or chord type. Then we weighted against common chords using the formula below,

$$weight(c) = \frac{i}{p(c) + \epsilon}$$

where ϵ controls the degree of weighting. We have tested ϵ using random float generator, and experiments showed that 0.05 - 0.10 is a good range for ϵ.

6.2 Evaluation Metrics

Considering composition as a creative process, we realized that evaluating the performance of chord generation model is a challenging task. We first used a strict metric, which is only counting point-wise matching as a correct prediction. Due to its objectiveness, we selected point-wise matching as our primary metric. However, since there are many reasonable chord choices for a given melody, the metric point-wise matching will not be able to completely reflect the performance of the model.

We then came up with our secondary metric, harmonious estimation, which includes harmonious outputs as correct predictions. For example, if the difference between the predicted note and the actual note is a major/minor third interval or a fifth interval, considering harmonic rules, we counted it as a reasonable answer. Such metric significantly improves the accuracy of our model. However, it might too loose to overlook uncommon chord progressions, for example, sixth-fourth chords.

We also planned to evaluate whether chord progressions follow harmonic rules. However, since there are no absolute harmonic rules in music theory, especially considering the various genres in our dataset, we thought using the criteria of pop music to evaluate chord progressions for jazz song is not appropriate. Thus, we discarded this metric.

6.3 Results and Analysis

Table 1 lists the results evaluated with point-wise matchings of chord roots from models of three settings. It turns out a unidirectional LSTM with trimmed melody input achieved the high accuracy and overfitted less. It verifies our observation that the bar-initial notes are more important to chord selection. The Bidirectional model achieved a slightly lower accuracy, but mainly because we test it few times due to its high computational cost. According to our training record, most good

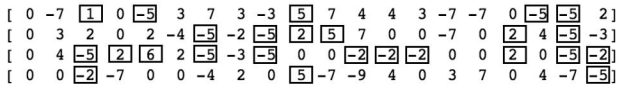
Table 1: Model performance, evaluated with point-wise matching

	train (72,418 bars)	test (17768 bars)	Prev-embedding	Melody-embedding
BLSTM, trimmed	33.0	30.5	18	10
LSTM, trimmed	**33.6**	**31.2**	30	20
LSTM, padded	31.1	not tested	5	5

models have bigger hidden dimensions for prev-LSTM than for melody-LSTM. It shows that our models need more computations to understand our representation of the chord progression.

The trimmed LSTM model achieved a harmony estimation accuracy of 60.5%, which represent 60.5% of the predicted root notes can most likely be considered well-harmonized. Figure 1 samples the difference between predicted and labeled chord roots. This figures circled the less-ideal intervals and showed that most predicted root notes are most likely well-harmonized.

Figure 4: Predicted root notes subtracting the labeled truth notes



In contrast, our model performed less ideally on non-root notes, or chord types. It seems that the accuracy on the second and third note, 0.69 and 0.90 respectively, are high. Since most chords in the data are major chords, if the model overwhelmingly outputs major chords ([root, 4, 7]), it would still achieve similar accuracies. This imbalance of chord types did initially cause the model to predict all major chords ([root, 4, 7]). After we introduced weighted loss, the model started to predict non-major chords, but mostly in wrong places. The various key signature the songs in this datasets have may have caused this limitation of our model, since multiple keys complicate the relation between notes and chord types.

Another limitation is that our model can barely end a melody sentence with a resolution (in C Major, a resolution could be a C major chord), so the predicted melodies usually end with a suspension, as the demo ("Guo_Ji_Ge_predicted.mp3") in our GitHub repository shows. This is a tougher problem to address, since it would require the model to learn the concept of sentences in music.

7 Conclusion and Future Work

Media often portrait recent progress of AI in art and music as a competition between the creativity of humans and machines. Our project, an automatic chord arrangement tool, aims to take a detour from this competitive narrative, and to create a helpful tool for music professionals and learners to explore their creativity.

In this paper, we experimented two RNN models, which generates chord progressions according to input melodies. In our second model, we tried to imitate musicians' thought process of chord arrangement and obtained some working predictions, especially for root notes.

As an immediate next step, we will try to address the problems in chord type predictions by expanding our dataset, and normalizing them all to C major to help the model understand the relation between notes and chord types. Also, we will design more sophisticated loss functions to give credit to non-exact working chord predictions, instead of treating them the same as badly-harmonized chords. We will address the problem of music sentence with a more comprehensive model that process a song as an entirety, instead of processing the information of only one bar.

This project is inspired by an idea of a music education application that helps music learners to compose and arrange in diverse genres, so this automatic chord arrangement tool is only the preliminary stage of our plan. We hope to continue exploring various means AI can assist our creativity as musicians.

8 Contributions

We worked together in data collection, reviewing previous literature, and writing the reports. Yulou was responsible for setting up and implementing different models using PyTorch. Shuxin focused more on data pre-processing, training models, and refining evaluation metrics.

References

[1] C. Chuan, and E. Chew, "A Hybrid System for Automatic Generation of StyleSpecific Accompaniment," 2011, http://axon.cs.byu.edu/~dan/673/papers/chuan.pdf

[2] M. Yang, W Hsu, and N. Huang, "Melody-to-Chord using paired model and multi-task learning language modeling," 2017, https://web.stanford.edu/class/cs224n/reports/2742800.pdf

[3] H. Lim, S. Rhyu, and K. Lee, "Chord Generation from Symbolic Melody Using BLSTM Networks," 2017, https://arxiv.org/ftp/arxiv/papers/1712/1712.01011.pdf

[4] PyTorch, "Sequence Models and Long-Short Term Memory Networks," Accessed May 19, 2018, https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

[5] Y. Lin, "LSTM-based Models for Sentence Classification in PyTorch," GitHub Repository, Accessed Jun 10, 2018, https://github.com/yuchenlin/lstm_sentence_classifier#lstm-based-models-for-sentence-classification-in-pytorch

[6] Music and Audio Research Group, "CSV Leadsheet Database," Seoul National University, Accessed Jun 10, 2018, http://marg.snu.ac.kr/chord_generation/