Automatic Harmonization using Recurrent Neural Networks

Oriol Barbany*, Natalia Gullon† and Sophia Kypraiou‡
Machine Learning (CS-433), School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne
Email: *oriol.barbanymayor@epfl.ch, †natalia.gullonaltes@epfl.ch, ‡sofia.kypraiou@epfl.ch

***Abstract*—The time series problem has been completely leveraged by recurrent neural networks and their variants, making them lead to very good results in fields like speech synthesis and natural language processing. This paper applies this architecture to the music field with the aim of using the Annotated Beethoven Corpus (ABC) to learn the underlying structure of the chord sequences in Beethoven's string quartets. Our model is able to predict the chords that follow a given sequence of arbitrary length provided by the user and it also offers the option to condition the predicted chords by some features like the global key or the length of the phrases.**

***Index Terms*—Recurrent Neural Networks, Music Processing, Time series**

I. Introduction

This work centers on exploiting the possibilities of the standardized and computer-readable way of labelling chords with harmonic analysis symbols used at the Digital and Cognitive Musicology Laboratory of EPFL to create the Annotated Beethoven Corpus (ABC) [1], which consists of Beethoven's string quartets provided with experts' harmonic analyses. We present a model for Automatic Harmonization, which was motivated by the fact that chords in classical musical are most of the time predictive and the same patterns are repeated in several occasions (plagal cadence, perfect authentic cadence, etc.). The model is based in Recurrent Neural Networks (RNNs), which are feed by a non-linear combination of both the previous chords and the features acting as conditioners of the future predictions. Another part of the presented model is dedicated on embedding the chords into a meaningful dimension based on the musical temperament before combining them with the conditioners. In the following sections, this representation is discussed and all the steps that lead to the final model are explained.

II. Methodology

A. Data cleaning

The dataset of study consists of the harmonic analyses of the chords in Beethoven's string quartets. Opposite to other similar time series problems where we have equally spaced temporal features like in speech synthesis, the chords can be defined with different temporal separation as depicted in figure 1. Moreover we also front both short-term and long-term dependencies, where these values can range from a few chords to some thousands. With special emphasis of this very long term dependency, in classical music it is very usual that patterns exposed in along other movements are reused and thus certain chords are influencing others of a distant future.

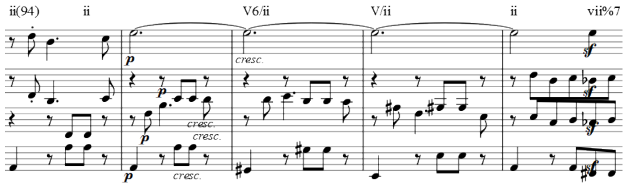

Fig. 1. Extract of the third movement of the String Quartet No. 8 in E minor

The smallest natural sequence division in the dataset is labelled by a special feature which indicates where the phrase ends are. These could be think analogously as any phrase in natural language, where we have a dot to indicate the endings. In the dataset of interest, we have a certain sequence of chord that can be considered as a phrase or not depending on the annotator criterion and thus ambiguity is much higher.

Even with the splitting by phrases, the smallest and presumably independent sequence of chords, we find very different lengths in our dataset (see figure 2), and the phrases can be very long, which is not desirable in a framework with RNNs. On the one hand, it is convenient to have sequences with the same length to train a RNN using several sequences at the same time to speed up this process, as well as to have losses in the same order of magnitudes. On the other hand, adding a padding to every phrase to fit the maximum length would face exploding and vanishing gradient problems, which is tackled by truncating the sequences. The following sections discuss two of the approaches that we take to handle the data preparation. In later sections, the results obtained with both will be compared.

Aside of splitting the dataset into sequences which can be used to train a RNN, the pre-processing of the data is also centered on cleaning the chords. Chords are provided as regular expressions, where not only the chord itself is represented but also additional information such as the aforementioned end of the phrases, the global key, the presence or absence of a pedal, etc. The unique number of chords in this case goes over one thousand, which is not very representative of the distinct chords in the standard Roman numeral notation, the internationally most common music theoretical notation system for harmonic analysis [2]. Therefore, the features

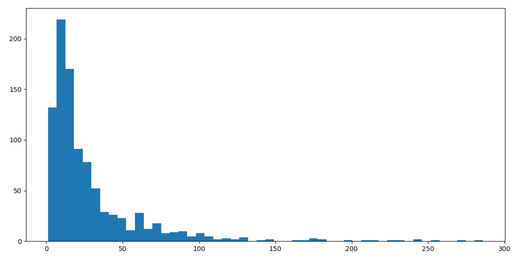

Fig. 2. Histogram of the phrase lengths

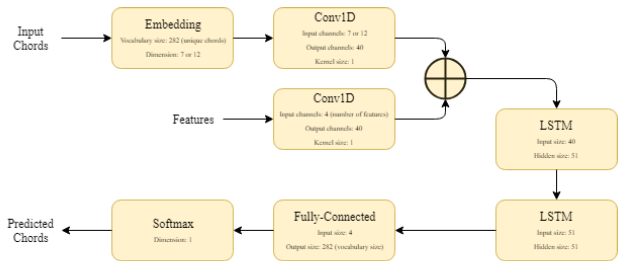

Fig. 3. Diagram of the conditioned model

conditioning the model are also used to remove the redundant information of the chord and hence decrease the vocabulary size, i.e. the number of unique symbols, to some hundreds depending on the considered features.

1) Sequential split: The data is split sequentially, which means that it is taken as it appears in the database and, therefore, it is not randomized. The partitions for train, validation and test are divided according to the their percentages, but the number of samples is rounded taking into account to not split the same movement into two different partitions.

2) Randomized split by phrases: The data is split by phrases and then randomized. The random sorted phrases are divided in train, validation and test datasets taking into account the different lengths of each phrase.

B. Model

The core of the presented model is the RNN, which we implement with a Long Short-Term Memory (LSTM) [3] using the Pytorch library [4]. We choose this model because as discussed above, the chord sequences have both short and long term dependencies. More concretely, we use the LSTMCell variant of Pytorch, which only implements one cell of an LSTM to ease the prediction of sequences with arbitrary length. This baseline model is an adaptation of a RNN used to predict the samples of a sinusoidal signal[1].

In order to fasten the training of the LSTM, we used the ground truth labels as input samples instead of using the predicted ones as when evaluating. This approach is known as professor forcing [5].

[1]https://github.com/pytorch/examples/tree/master/time_sequence_prediction

In the baseline model, we simply feed the cleaned chords into the RNN, but in a more advanced approach, we add an embedding layer of dimension 7 or 12. The idea of choosing an embedding dimension consistent with the current musical representation of the tempered system is inspired by the paper Neural Discrete Representation Learning [6], where a fully unsupervised model learned high-level speech descriptors that were closely related to phonemes.

In order to condition the model, we combine the features with a 1-dimensional Convolutional Neural Network (CNN) of unitary kernel, which acts a linear layer but accepts more dimensions into it [7]. This provides a non-linear combination of the features that successfully conditions the outputs RNN. This idea is inspired by the Text-To-Speech system presented in [8]. The full model as well as its parameters are depicted in figure 3.

C. Cross-validation

In this section, the results for the different configurations of our model as well as for some relevant hyper-parameters are shown.

JET	With mass	Best degree	Best λ	Accuracy
0	True	7	10^{-5}	0.797 ± 0.004
0	False	12	10^{-5}	0.986 ± 0.003
1	True	9	10^{-4}	0.739 ± 0.005
1	False	10	10^{-6}	0.977 ± 0.007
2	True	8	10^{-5}	0.557 ± 0.006
2	False	9	10^{-10}	0.892 ± 0.016
3	True	9	10^{-4}	0.832 ± 0.008
3	False	9	10^{-2}	0.975 ± 0.011

TABLE I
BEST PARAMETERS OBTAINED WITH CROSS-VALIDATION

III. EXPERIMENTS AND RESULTS

In Table II we can see the different performances of some of the models tested. We added a random guess model as a reference to see the improvement of each model. All those models have a previous data cleaning that only consisted in removing the features with meaning-less values.

– **Model A:** Random guess (reference)
– **Model B:** Gradient Descent (MSE loss function)
– **Model C:** Stochastic Gradient Descent (MSE loss function)
– **Model D:** Least squares
– **Model E:** Ridge regression
– **Model F:** Regularized logistic regression
– **Model G:** Regularized logistic regression with JET and mass filtering and adaptive learning rate

As we can see, the best model in terms of the obtained accuracy, is the regularized logistic regression with Gradient Descent optimization and adaptive learning rate. Regarding the approach to update the learning rate, we decay by a γ factor after a given number of iterations. This is inspired by the Step Learning rate Scheduler of Pytorch[4].

The decaying factor of the learning rate is set to $\gamma = 0.1$, which is the default value provided by the Pytorch library, and

Model	Accuracy
A	0.500 ± 0.001
B	0.706 ± 0.002
C	0.709 ± 0.003
D	0.774 ± 0.001
E	0.767 ± 0.001
F	0.743 ± 0.001
G	0.823 ± 0.004

TABLE II
PERFORMANCE WITH DIFFERENT MODELS

it is updated every 2000 iterations. This last value is set based on the evolution of the loss function across iterations to avoid having a very large step size at the first iterations or very few improvements in the last ones.

IV. Acknowledgements

This research was promoted by Fabian C. Moss, PhD candidate at the Digital and Cognitive Musicology Laboratory of the Digital Humanities department at EPFL (https://dhlab.epfl.ch/).

V. Conclusions

The source code of this project is available on GitHub[2].

VI. Future work

References

[1] M. F. Neuwirth M, Harasim D and R. M, "The annotated beethoven corpus (abc): A dataset of harmonic analyses of all beethoven string quartets," 2018, front. Digit. Humanit. 5:16. doi: 10.3389/fdigh.2018.00016. [Online]. Available: https://github.com/DCMLab/ABC

[2] E. Aldwell and A. Cadwallader, *Harmony and Voice Leading*. Cengage Learning, Jan. 2018, google-Books-ID: T69EDwAAQBAJ.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[4] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[5] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor Forcing: A New Algorithm for Training Recurrent Networks," *NIPS*, 2016. [Online]. Available: http://arxiv.org/abs/1610.09038

[6] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural Discrete Representation Learning," in *NIPS*, 2017. [Online]. Available: http://arxiv.org/abs/1711.00937

[7] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[8] O. Barbany, A. Bonafonte, and S. Pascual, "Multi-Speaker Neural Vocoder," *IberSpeech*, 2018. [Online]. Available: http://iberspeech2018.talp.cat/download/IberSPEECH_2018-Proceedings.pdf

[2] https://github.com/Barbany/ML-Project2