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*—Over the past years, 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 aimed 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, Computational Models of Music**

I. Introduction

This work focuses 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 analysis. We present a model for Automatic Harmonization, which is motivated by the fact that chords in classical music 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 on Recurrent Neural Networks (RNNs), which are feed by a 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 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

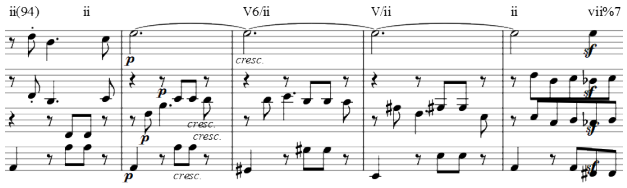

Fig. 1. Extract of the third movement of the String Quartet No. 8 in E minor

patterns exposed along with other movements are reused and thus certain chords are influencing others of a distant future.

The smallest natural sequence division in the dataset is labelled by a special feature which indicates where the phrase-ends are. This can be thought 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 chords that can be considered as a phrase or not depending on the annotator criterion. Therefore, ambiguity is much higher compared to dots delimiting sentences.

Even with the splitting by phrases, the smallest and presumably independent sequence of chords, we find very different lengths in our dataset (see Figure 3), as well as very long phrases, 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 the process, as well as to have losses in the same order of magnitude. On the other hand, adding a padding to every phrase to fit the maximum length would face exploding and vanishing gradient problems due to training with very long sequences. This is usually tackled by truncating the sequences, as we propose. The following sections discuss two of the approaches that we implement 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 previously mentioned end of 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,

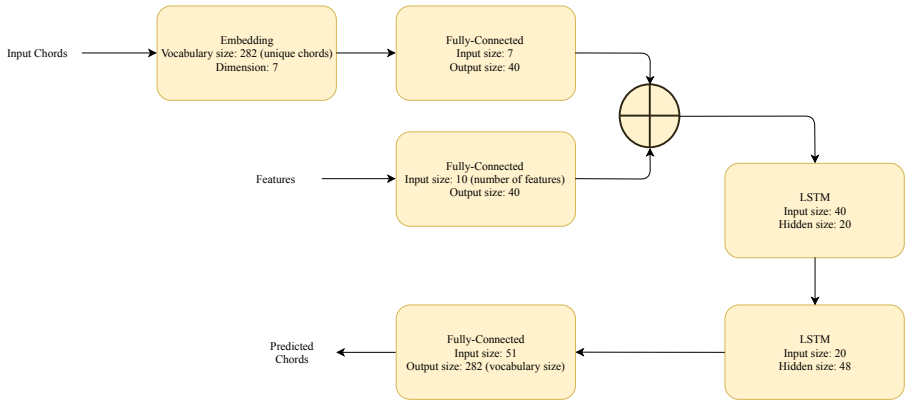

Fig. 2. Diagram of the conditioned model with improved hyper-parameters

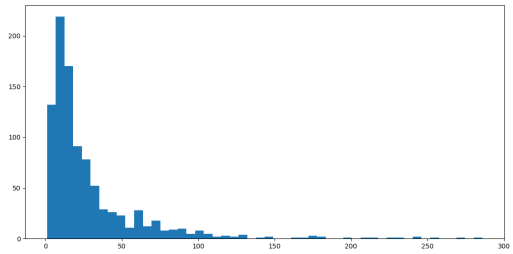

Fig. 3. Histogram of the phrase lengths

the internationally most common music theoretical notation system for harmonic analysis [2]. Therefore, the features 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.

As mentioned before, we propose two different approaches to handle the data before training. We should mention that in both approaches, we truncate the total length so that each of the sequences assigned to every partition is complete. This means that we don't need padding and hence an additional class does not have to be added. This is motivated as it does not make sense to only have the padding symbol appearing on one unique sequence on every partition because the saturation required by this symbol is hardly learned with only one example. The two different approaches for data handling are:

1) Sequential split: This first approach is based on sequentially splitting the data, which means that the data is taken with the same order as it appears in the database and, therefore, it is not randomized. The partitions for train, validation and test are divided according to the percentages (80%, 10%, 10% respectively), but the number of samples is rounded considering to not split the same movement into two different partitions. Note that the test partition is not aimed as a final result but as to measure the quality of predictions on unseen data. While we use the training partition to train the model and the validation to tune the hyper-parameters as usual, our objective is to predict sequences with any starting chords and features, and in this case, we don't have a ground truth to compare with.

2) Randomized split by phrases: In this second approach, the data is split by phrases and then randomized. The randomly sorted phrases are divided in train, validation and test datasets with the same percentages as before, 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].

In the baseline model, we simply feed the cleaned chords into the RNN, but in a more sophisticated approach, we add an embedding layer of dimension 7 (i.e. different notes per octave in the Heptatonic scale, which is the most common in modern Western music) or 12 (i.e. different notes per octave on the Chromatic scale). 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.

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

In order to condition the model, we combine both the chords and the features by feeding them to independent fully-connected linear layers that has matching output dimension. With this setup, we can sum the mappings of the chords and the features before feeding this tensor to the RNN. This idea is inspired by the Text-To-Speech system presented in [7], where they do the same to condition the audio samples on the acoustic features and add an embedding representing the speaker. In this paper, this mapping is implemented with a 1-dimensional Convolutional Neural Network (CNN) of unitary kernel, but it acts as a linear layer with the only difference that it accepts more dimensions into it [8]. Our inputs (input chords and features shown in Figure 2) only have two dimensions; the batch and the input size (i.e. number of features, dimension of embedding or simple scalar depending on the model) and hence it is possible to implement it with a simple linear layer. This provides a combination of the features that successfully conditions the outputs RNN. The full model as well as its parameters are depicted in Figure 2.

C. Cross-validation

In order to tune the hidden sizes of the two LSTM used in our model, we test several hyper-parameter combinations and choose the one yielding a lower Cross Entropy Loss. The hyper-parameters are shown in the diagram of Figure 2. Note that this tuning was only done for the best model (see Table I). While testing each of the models, all the hyper-parameters and the data are exactly the same to provide a fair comparison.

III. Experiments and results

Our models are first tested using sequences of length 50 samples for both approaches. We choose this value as a trade-off between having a small value so that the RNN is able to properly learn the sequences and a large value so that the network is able to see the short and long-term dependencies usually present in classical music, as already explained in II-A. To check proof that our model is not training well due to sequences that are too long and hence are potential of generating vanishing gradients during the back-propagation through time, we also test the model with sequences of 30. It ended up with roughly the same results with respect to the loss on the validation partition.

We use 3 features that condition the model and help improving the prediction as seen in Table I in the comparison of all the models. These features are the global and local key and the phrase-end. For the best model, we also try picking more features, which turn out to improve even more the loss. Apart from the before mentioned features, we also use the *altchord* (alternative analyses of the chords), the length of the chord, the pedal, the form, figured bass, changes and relative root. Basically, these are all the features that doesn't refer to very specific ones such as the movement or the opus, which could help in prediction but are not of interest for the purpose of this research. The extra features considered here, are mostly sparse, and that's why they were not included from the very beggining. Moreover, most of them are too specific and it can be discussed whether it makes sense or not depending on the level of specificity desired of the features conditioning the harmonizer.

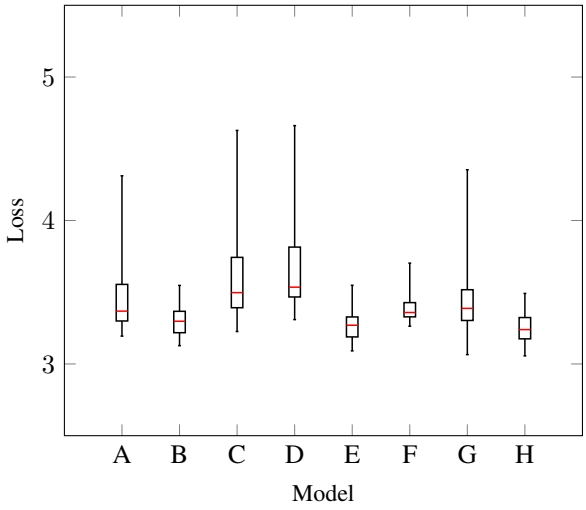

Fig. 4. Boxplots comparing models' performance

Regarding the hidden sizes of each of the LSTM modules, we do the comparison of all the models with a hidden size for both LSTMs of 51. This result is taken from the baseline model for predicting sinusoidal signals mentioned above that we adapted for our task. Nevertheless, after cross validation trying several hidden sizes, the setup that works better for our application is to have a hidden size of 20 for the first LSTM and 48 for the second one.

In addition to the comparison of test loss for all the models described in Table I, boxplots showing the variability of the test loss for each model are also presented in Figure 4.

We can see that we get better results with the randomized split by phrase approach, explained in section II-A, for all the configurations tested. We should also note that using an embedding of size 7 leads to better results in both approaches. Moreover, we can find some meaningful representation with the embeddings of this size as seen in the snapshot of Figure 5. Note that the proximity of the elements to the observer is shown by the size of the chord numeral, and most of the nearer ones are variants of the seventh chord, i.e. `VII`.

Having a deeper look at the results for that randomized split by phrase approach, we can also observe that adding the embedding and conditions to our model improve the results, as expected. Therefore, the best results are obtained with model I (conditioned model, with randomized split by phrase approach and with an embedding size of 7). The loss curves for the train and validation sets are presented in Figure 6.

Regarding the learning strategy, the chosen optimizer is Adaptive Moment Estimation (ADAM) [9]. ADAM is a stochastic gradient descent algorithm with adaptive learning rate that not only changes the weights following the direction based on the gradient of the loss function, which in our case is the Cross Entropy loss, with respect to the parameters, but

Model	Architecture	Data cleaning approach	Embedding size	Cross-Entropy Loss
A	Chords	Sequential	None	3.4885 ± 0.3073
B		Randomized	None	3.3026 ± 0.1129
C	Embedded chords	Sequential	7	3.6213 ± 0.3801
D			12	3.6837 ± 0.3698
E		Randomized	7	3.2778 ± 0.1227
F			12	3.4004 ± 0.1177
H	Conditioned model	Sequential	7	3.4263 ± 0.1690
I		Randomized	7	3.2453 ± 0.1193

TABLE I
LOSS VALUES OBTAINED FOR EVERY MODEL

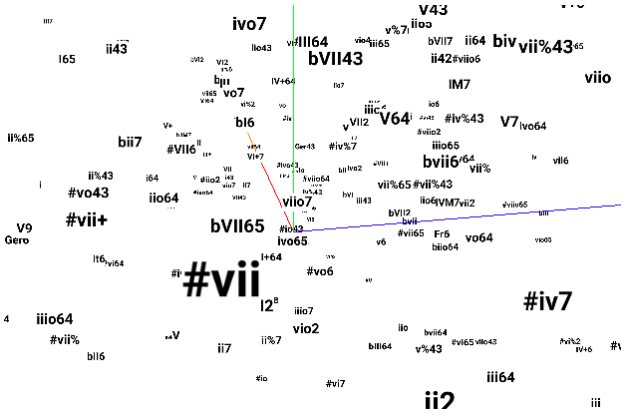

Fig. 5. Projection of the 7-dimensional embedding into 3 dimensions using Principal Component Analysis

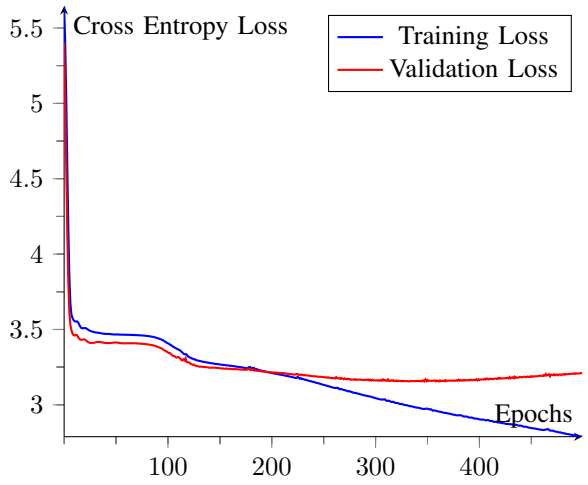

Fig. 6. Loss curve for the best model (I)

also based on the values of the previous updates. The learning rate is set to 0.01 and the other parameters that define ADAM are chosen as the default ones in the Pytorch implementation. A learning rate controller is also implemented using a cosine annealing schedule [10]. In this case, learning rate is defined with a cosine and thus it has a certain periodicity of going up and down. The periodicity of the learning rates given by cosine annealing is set to 50.

The addition of gradient clipping tries to solve the already mentioned the typical problems faced with RNNs and stated to tackle them by truncating the inputs and using LSTMs. To prevent exploding gradients, we set the maximum allowed value of the gradients to 0.5, so they are clipped in the range [-0.5, 0.5].

Finally, our final model achieves a loss of 3.0514 ± 0.2556 (see Table I to see improvement of the model). For this last, we also compare the accuracy, which turn out to be very low and around 25% for this optimal setup. Although the seemingly bad result for this last metric, we think that accuracy is not representative of our problem given that we have a lot of different chords and the distances between them are not the same. Think for example in the difference between `I` and `VII`, which have all the different notes, or `I` and `I6`, which have exactly the same notes but in different ordering.

Regarding the predictions, we can find that the algorithm successfully learned some of the most used transitions as `I - V`, but for some sequences that are very long, some chords saturate, meaning that they appear several times consecutively. However, when improving the results with respect to the test loss, the predictions are even more saturated (all the predictions are the same chord). That leads to the conclusion that the loss used is not very representative for our problem, so we should find another that fits better.

IV. CONCLUSIONS

This model automatizes the task of harmonization, which can be used to help composing new music with Beethoven's style of writing string quartets. Given a melody, one can use the chord sequences to add more instruments to create a more elaborated musical work. This is enhanced by the option to condition the model, so one can choose parameters such as both the local and global key of the sequence, the phrase endings, etc.

The main problem encountered in this project is that training a RNN for some task with a large vocabulary size like in our case, we need huge amounts of data to achieve higher accuracies. This dataset is singular in its kind and it's very costly to create new data similar to it. That's why we propose a new data cleaning approach, based in taking overlapping sequences, so we can train the model over more data and start sequences from a wider variety of chords.

In order to compare our model, we also consider appropriate to reconsider which loss function fits better to this problem than the accuracy.

The conditioning forces the RNN to predict chords given some constraints, but it can be argued that parameters such as the phrase ending can easily not be successfully used because a phrase cannot end at every point. In this setup, it is very natural to think that this feature as well as others, like the change of key, have a smooth transition on the real chord sequences. This means that the model should be able to not only see the next features but also further in the future. This approach presented in [7] has already been proven for Text-To-Speech, which as well as in our model, the future features that will condition the model are known beforehand.

Finally, to promote the development of this project as well as the use of the Annotated Beethoven Corpus, which is already available in the repository of the Digital and Cognitive Musicology Laboratory of EPFL[2], the source code of this project is available on GitHub[3].

V. Acknowledgements

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

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

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

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[10] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, vol. abs/1608.03983, 2016. [Online]. Available: http://arxiv.org/abs/1608.03983

[2] https://github.com/DCMLab/ABC

[3] https://github.com/Barbany/Automatic-Harmonization