Score and Performance Features for Rendering Expressive Music Performances

Dasaem Jeong, Taegyun Kwon, Yoojin Kim, Juhan Nam

Graduate School of Culture Technology, KAIST, Korea
{jdasam, ilcobo2, luciaicul, juhannam}@kaist.ac.kr

May 2019

1 Introduction

Rendering a human-like expressive music performance is a challenging task for computers. One approach is to train a learning model to imitate human performances given music scores. This requires the model to take the score data as if human performers read and understand the music score. In addition, the performance data aligned with the score should be represented to render the human-like performance effectively.

We recently proposed a neural-network-based system that renders expressive piano performances from music scores[2]. To implement the performance rendering system, we numerically extract features from score data in MusicXML and performance data in MIDI. In this paper, we present the details of feature extraction from the score and performance data. Though these features are originally devised for the performance rendering system, they can be also used for other tasks such as chord recognition, composer identification, or performance evaluation.

2 Score Features

Our rendering model takes a sequence of notes as an input. Therefore the score and performance information should be encoded in note-level. Our goal is to encode the score information into note-level features so that our network model can "understand" characteristics of each note in the context of the entire score.

Because of computational limitation on memory usage, we cannot use a full sequence of notes from the entire piece as an input. Therefore, the piece has to be sliced into several mini sequences. The problem with this process is that many of important directions such as tempo markings (e.g. *Allegro, Adagio, più mosso*) or dynamic markings (e.g. *pp, mf*) that precede the notes can be

lost during the slicing. To solve this problem, we have to embed the expression markings into note-level score features.

Many of previous works employed not only the basic features such as pitch and duration of the note but also higher-level features such as key and metric information. [1]. In addition to them, we use more detailed information such as the duration of following rest, articulation markings (e.g. trill, staccato, tenuto), distance from the closest preceding tempo and dynamic directions, slur and beam status.

To embed various directions into note-level score features, we have to know the effective range of each direction. We define the end position of each direction with a simple categorical rule. We categorize directions into three hierarchy levels. The effective range of each direction is from its start position to appearance of other direction in the same or higher level of the hierarchy. The high-level of directions consists of direction with absolute meaning such as *Allegro, Adagio, Largo* in tempo marking and and dynamic markings like ***pp***, ***mf***. The mid-level directions include tempo markings like *meno mosso, più animato* and dynamic markings like *rinforzando*. The low-level directions show a local change in tempo or dynamic, such as *ritardando, accelerando* or crescendo, sforzando.

Another issue is how to embed these directional words into numeric values. One of the solutions is to represent them as a long one-hot vector so that each element of the vector represents a single word using a dictionary that includes all possible directions. The problem is that there are a variety of possible musical directions and the appearance of individual direction is extremely sparse in general. Therefore, we embed the directions into a vector with numeric values based on our domain knowledge. For example, a tempo vector has three elements, and each element represents absolute tempo, relative tempo change, and acceleration, respectively. All detailed implementation and discussion on the limitation of these rules will be covered in the full paper.

3 Performance Features

Human performance on the piano is captured well in MIDI format. To train the performance rendering model, we first align notes in the score to the performance MIDI using an automatic alignment tool [3] and then extract tempo, onset deviation, velocity, articulation, and pedal as note-level performance features. The tempo represents the number of beats played in a minute. The beat is defined by the time signature of the measure. For each beat in a piece, we calculate the time interval between two beats in the performance. The onset deviation is a feature that explains a micro-timing or asynchronous of performance, or tempo rubato within a single beat. For each note, we calculate how the note onset is displaced in its 'in-tempo' position based on the pre-defined tempo. The dynamics of performance can be represented with MIDI velocities of the corresponding notes.

While these features have been widely used in many of previous works [1], we also tried to model the use of piano pedals, which is one of the most challenging

problems in the performance rendering. We encoded the pedal information in performance MIDI into note-level features by observing the pedal state in four different positions of the note: start of the note, offset of the note, minimum pedal value between note onset and offset, minimum pedal value between note offset and the following new onset.

4 Result and Conclusion

We have trained an RNN-based system with the proposed feature scheme. The result of how the model generated different performance based on musical directions in input MusicXML is presented in Figure 1. The result showed that our moddel could apply musical directions while generating the performances. A listening test showed that our model successfully achieved human-like expressions. The performance generated by our model is uploaded in YouTube[1].

We release our Python-based feature extraction for MusicXML as a publicly available software library[2], hoping that it can be useful for various music processing tasks.

References

[1] Carlos Eduardo Cancino-Chacón, Maarten Grachten, Werner Goebl, and Gerhard Widmer. Computational models of expressive music performance: A comprehensive and critical review. *Frontiers in Digital Humanities*, 5:25, 2018.

[2] Dasaem Jeong, Taegyun Kwon, and Juhan Nam. VirtuosoNet: A hierarchical attention RNN for generating expressive piano performance from music score. In *NeurIPS 2018 Workshop on Machine Learning for Creativity and Design*, 2018.

[3] Eita Nakamura, Kazuyoshi Yoshii, and Haruhiro Katayose. Performance error detection and post-processing for fast and accurate symbolic music alignment. In *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[1] `https://www.youtube.com/watch?v=hPBR2Rxu3-s&list=PLkIVXCxCZ08rD1PXbrb0KNOSYVh5Pvg-c`
[2] `https://github.com/jdasam/pyScoreParser`

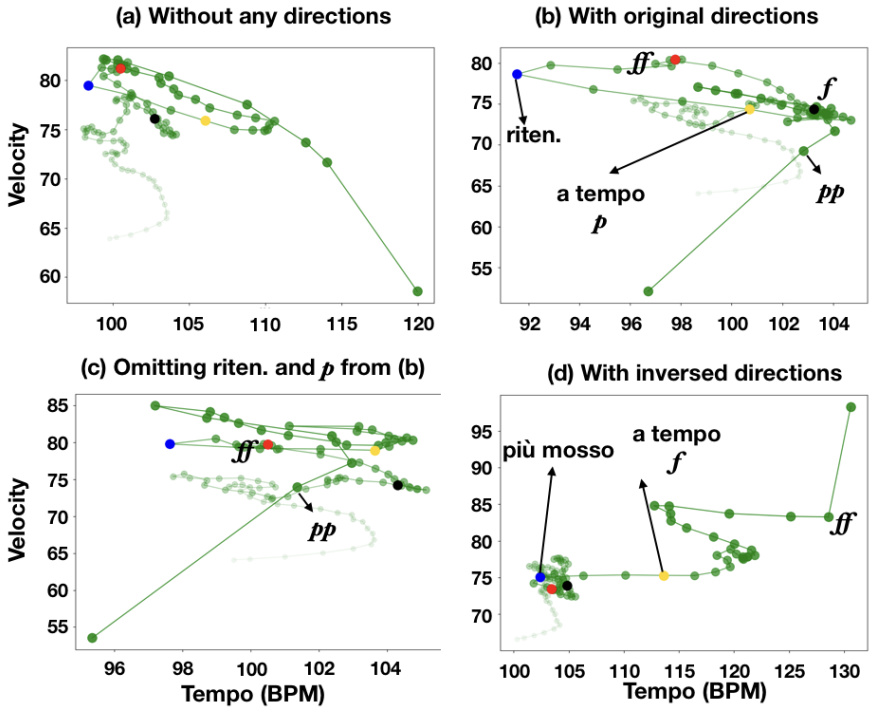

Figure 1: The visualization of generated MIDI performance by our trained model depends on directions in input MusicXML. The test piece is Étude op. 25 No. 9 by Chopin. Each point represents a tempo and average dynamics of a beat in the piece, which were smoothed with previous values. The smallest and most transparent point represents the beginning beat of the piece and the largest and least transparent point represents the end beat. Black, red, blue, and gold colored point represent first beat of measure 25, 33, 36, and measure 37's second beat, respectively. In (a), we deleted every tempo and dynamic markings in the input MusicXML. Since our model was trained from other various classical music, it generated tempo and dynamic change from notes' pitch and rhythm. In (b), the input has the tempo and dynamic markings from original music. As shown in the figure, The performance had clear dynamics highlight when ***ff*** appeared. Also, because of the ***pp*** direction right before the last measure, there was a clear *ritardando* at the end of the piece. When we omitted a *ritenuto* and ***p*** at measure 36 and 37 as presented in (c), the ***ff*** of measure 33 affected until ***pp*** appeared. When we changed *ritenuto* in measure 36 to *più mosso*, it was not cancelled by *a tempo* by our hierarchy definition, and the tempo has changed until the end of the piece

Appendices

A List of Features

The list of input features are: **a)** pitch in MIDI pitch(int) **b)** octave number (int) **c)** pitch class (12-D one-hot vector) **d)** duration **e)** pitch in octave (int)

and pitch class as one-hot vector **f)** measure length **g)** duration of following rest **h)** duration after the corresponding tempo marking **i)** duration after the corresponding dynamic marking **j)** relative position in measure [0 - 1] **k)** relative position in entire piece [0 - 1] **l)** if grace note, number of notes between itself and its following non-grace note, including itself (int) **m)** is preceded by a grace note (bool) **n)** is followed by a fermata rest (bool) **o)** time signature denominator vector (4-D vector: 2, 4, 8, 16) **p)** time signature numerator vector (5-D vector: duple, triple, quadruple, compound, and other) **q)** slur and beam status vector (6-D multi-hot vector: start, end, continue for slur and beam) **r)** composer vector (one-hot vector) **s)** notation marking vector for trill, fermata, accented, strong accented, staccato, tenuto, arpeggiated, cue, slash (8-D multi-hot vector) **t)** beat strength of the note (rule-based, float) **u)** tempo marking vector (rule-based, 5-D vector) **v)** dynamic marking vector (rule-based 4-D vector)

Every duration or length is measured in a quarter note. For example, a duration of an eight note is 0.5, and measure length of 4/4 bar is 4.

These global conditioning features are added to input features along with the corresponding score features: **a)** embedded tempo marking vector of the beginning of the piece **b)** tempo of the first ten beat in quarter note per minute (log) **c)** composer of the piece (one-hot vector)

The output features are: **a)** Tempo in quarter note per minute, calculated for every beat (log) **b)** MIDI velocity **c)** deviation of the note onset compared to 'in-tempo position' in quarter note **d)** articulation (log) **e)** value of sustain pedal at note onset [0 - 127] **f)** value of sustain pedal at note offset [0 - 127] **g)** smallest sustain pedal value between the note's onset and the offset [0 - 127] **h)** elapsed time between the note onset and the moment g) is detected **i)** smallest sustain pedal value between the note offset and its closest next note onset **j)** elapsed time between the note offset and the moment i) is detected **k)** value of soft pedal at note onset.

We used additional features for modeling a trill note. The trill features are **a)** number of trill notes per second **b)** relative velocity of the last trill note compared to the first trill note **c)** relative duration ratio of the first trill note **d)** relative duration ratio of the last trill note **e)** whether the trill starts with higher note (alternative trill).

A.0.1 Rule-based Embedding

The beat strength represents how much strong the beat is in the note position. The strength is set 4 if the note is in the start of the measure, 3 if the note is in half position of a duple meter, 2 if the note is in third or two third position of triple meter. Similarly, we defined the beat strength of quarter positions or eighth positions.

We made a conversion dictionary that takes tempo and dynamic markings and make it into five-dimensional vectors. Each dimension of the vector has a value between -1 to 1. Below are the examples of the conversion.

The last dimension of dynamic embedding vector represents number of unresolved crescendo or decrescendo between notes and corresponding dynamics.

Markings	Encoded Vector
Allegro	[0.5, 0, 0, 0, 0]
Andante	[-0.5, 0, 0, 0, 0]
Allegro - più mosso	[0.5, 0.6, 0, 0, 0]
Sehr rasch - etwas langsamer	[0.8, -0.3, 0, 0, 0]
Scherzo, Allegro	[0.5, 0, 0, 0, 1]
Andantino - rit	[-0.3, 0, -0.5, 0, 0]
Allegro molto - riten	[0.6, 0, 0, -0.5, 0]
Allegro molto - poco riten	[0.6, 0, 0, -0.3, 0]
mp - cresc	[-0.2, 0.7, 0, 0, 0]
f - fz	[0.4, 0, 0.3, 0, 0]
ff *- rinforzando molto*	[0.7, 0, 0, 0.7, 0]

Table 1: Examples of tempo and dynamic markings to the encoded vectors

A crescendo represented as a wedge has clear start and end position. Usually, a wedge is followed by dynamic markings like ***p*** and ***mf***, or followed by a decrescendo wedge. But sometimes, it can be followed by no other sign. We accumulate this unresolved crescendos and encode it to the individual notes.