Study and Application of Monte Carlo Algorithm for AI-Based Music Generation

Jun Min and Lei Wang(✉)

College of Electronic and Information Engineering, Tongji University, Shanghai, China
{minjun,wanglei}@tongji.edu.cn

Abstract. When generating music via algorithms, it is essential to extract music characteristics and the distribution of notes. A Monte Carlo simulation framework for music generation was proposed on the premise of maintaining the authenticity of music samples. Those samples, which are stored in MIDI format, were first converted into data that can be processed by computes. To simulate music time series, we adopted Logistic regression. Except for time series, the converted data also include three parameters: duration, pitch, and velocity. We first solved the correlation coefficient matrix and standard deviation of the three parameters, and then analyzed them using Monte Carlo method and summarized their distribution patterns.

Keywords: Music Generation · Monte Carlo Method · Data Analysis · MIDI

1 Introduction

Researchers have actively explored the use of artificial intelligence (AI) algorithms in music composition. Currently, Google and Sony are also engaged in active research in this area. The recent work of Magenta, an open-source project developed by researchers and engineers from the Google Brain team, were based on a 90-min melody created by Tensorflow (ISMIR2010) [1]. Sony's AI-based Flow Machines can learn various musical styles from a huge library of music, and generate music via style-shifting, optimization and interaction techniques [2]. Moreover, the development of neural networks, such as Support Vector Machines (SVMs) (ACMMM2016) [3], Deep Neural Networks (DNNs) (ICMM2019) [4], Convolutional Neural Networks (CNNs) (ICASSP2019) [5]. and Recurrent Neural Networks (RNNs) [6], has also been shown to be effective for the processing and generation of music files. As theories and technologies continue to advance, more and more new methods are being used to process or generate new music. Driven by the development of DNNs, other neutral networks including SVMs, DNNs, CNNs, and especially RNNs, have again attracted researchers' interest. Long short-term memory (LSTM), an artificial RNN architecture, is a commonly used for music processing. It is a special form of RNNs [7]. With Forget gate, it can decide whether to keep the information from the previous timestamp or forget it. It can also process sequences of data by studying how to control a particular point of time. This approach, however, fails to handle the long sequences needed for generating music, since it is not yet able

C. F. Peng et al. (Eds.): EIMT 2022, AHSSEH 3, pp. 392–402, 2023.
https://doi.org/10.2991/978-94-6463-012-1_44

to handle the long sequences required. Thus, the outputs are often unsatisfactory if we generate music only through LSTM. To address this issue, the most common approach is to superpose multiple neural networks into one and optimize them based on LSTM.

Before using artificial neural networks (ANNs) to generate music, an interesting approach is to learn the probability distribution of notes. The method models the music structure as a probability distribution, which means that the probability of each note appearing in a particular part and in a particular key, while the probability of the current note must depend on the distribution of notes at the previous moment. From the perspective of data analysis methods, the processing of music data is generally divided into two directions. One is to select a music file, and analyze the music spectrum and waveform using computers. In this way, music characteristics can be obtained, helping machines to capture music information and learn how to compose music [8]. Another approach is to analyze a large amount of MIDI file using computers that will then extract musical structure and form an empirical model to store it [9].

The structure of RNNs is well suited for this application, as traditional ANN algorithms require time series for modeling. Compared to CNNs, RNNs are able to maintain the persistence and continuity of information. This property allows the system to determine the possible distribution of the upcoming notes based on previous notes. However, the inherent defects of their structure have prevented them from obtaining notes far from the current position (EMNLP2014) [10]. Although the above methods can simulate and synthesize new sequences of notes, they all require that the data samples remain time-invariant. For example, when training data is moved forward or backward by several time steps, the model will not recognize them as different training data, but moves the training data up and down as a whole, while failing to maintain this invariance in pitch transfer, which increases time costs and places higher demands on system hardware [11]. Therefore, the relative positions between notes, rather than the absolute positions of the notes, must be considered when designing systems network architecture [12]. The Monte Carlo algorithm has a great advantage over most music modeling methods because it focuses only on the relative positions when processing music samples, not on the absolute positions [13]. Besides, music composition is usually inseparable from music theory and musical inspiration [14]. For the former, the data set can be trained by the ANNs, while the latter needs to summarize the influence of different factors to achieve breakthroughs and innovations, and this process is simulated here using Monte Carlo method.

2 Data Processing

The dataset we adopted is all from GiantMIDI-Piano, a classical piano MIDI dataset created by Qiuqiang Kong et al. in 2020. The sample data was processed into numbers using a conversion tool. Table 1 and Fig. 1 show the a five-line staff and converted parts of the famous piano music, *For Elise*.

The first column in Table 1 indicates the beat of notes (Quarter note); the second one represents the duration of notes; the third one represents the MIDI channel (1–16); the fourth one represents the pitch (range 0–127), and the fifth column represents the speed of keys.

Table 1. MIDI Messages

ONSET	DURATION	CHANNEL	PITCH	VELOCITY	ONSET	DURATION
0	0.575	12	76	31	0.0000	0.2875
0.5166	0.500	12	75	47	0.2583	0.2500
0.9500	0.4541	12	76	58	0.4750	0.2270
1.3250	0.5375	12	75	66	0.6625	0.2687
1.7500	0.4250	12	76	57	0.8750	0.2125
2.1333	0.4791	12	71	59	1.0666	0.2395
2.9541	0.5383	12	74	55	1.2687	0.2395

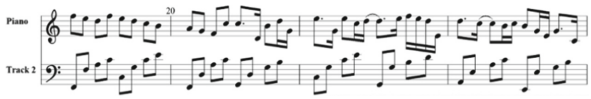

Fig. 1. Music score sample

The five-line staff is an important way to observe and record music. The original music can be well identified and reproduced with the help of five-line staff. MIDI files can be converted into data via a converter, as shown in Table 1. Therefore, using this approach to pre-process music data makes it easier to process music samples and learn how notes distribute, which paves the way for the system to generate new music samples via machine learning.

3 Markov Analysis

It is known that Markov chains can converge to stable distribution. If a Markov chain with state-transition matrix **P** of a music note is constructed, and the stable distribution is **P(x)**, then we get a note transfer sequence $\mathbf{x_1}$,……, $\mathbf{x_n}$, $\mathbf{x_{n+1}}$,……, by transferring from any initial state $\mathbf{x_0}$ along the Markov chain. If the Markov chain converges in step **n** and the output tends to be stable, then the samples after $\mathbf{x_n + 1}$ and $\mathbf{x_n + 1}$ must also satisfy the **P(x)** distribution, which satisfies the condition that the possible distribution of music samples are **P(x)**. However, in general the target stable distribution $\pi(x)$ and its state-transition matrix **Q** are not stably distributed. To reach stable distribution, let:

$$\pi(i)Q(i,j)\alpha(i,j) = \pi(j)\alpha(j,i) \tag{1}$$

In Eq. 1, $\pi(i)$ is the initial distribution of notes, $\pi(j)$ is the target stable distribution of notes, $Q(i,j)$ is the state-transition matrix of notes, and $\alpha(i,j)$ and $\alpha(j,i)$ are the acceptance rates of notes, which are usually between 0 and 1. In order to make both sides of Eq. (1) equal, the following measures are required, as in Eq. (2).

$$\begin{aligned} \alpha(i,j) &= \pi(j)Q(j,i) \\ \alpha(j,i) &= \pi(i)Q(i,j) \end{aligned} \tag{2}$$

Thus, the state-transition matrix turns into:

$$p(i,j) = \alpha(i,j) \times Q(i,j) \tag{3}$$

In Eq. (3), the target matrix $p(i,j)$ is obtained by multiplying the state-transition matrix **Q** of Markov chain by $\alpha(i,j)$. This method is extended referring to Rejection Sampling, which is used to obtain anomalous distributions with a common distribution by rejection rates. However, if the Markov chain remains constant during note sampling while using the Monte Carlo algorithm, the rejection rate will increase dramatically and the computation time will become very long. This is the reason why Markov chains set the rejection rate relatively small in the transmission process. This could help improve the operation efficiency and keep stable distribution. The acceptance rates $\alpha(i,j)$ and $\alpha(j,i)$ need to be scaled up in the same ratio so that one of them is scaled up to 1 first, thus increasing the acceptance rates during sampling. As in Eq. (4).

$$a(i,j) = \left\{ \frac{p(j)q(j,i)}{p(i)q(i,j)}, 1 \right\} \tag{4}$$

An important reason for using Markov chain is that no matter what the initial distribution of notes is, the system will eventually converge to a fixed value when the number of state transfers **n** is large enough, as long as the state-transition matrix is constant. Importantly, in order for a Markov chain to satisfy a stable distribution, the system also needs to satisfy the following conditions:

- The number of possible states is limited.
- The transition probabilities among various states needs to be fixed.
- The system cannot be a simple loop from **X** to **Y** and then from **Y** to **X**.

3.1 Solutions of State-Transition Matrix

Let the homogeneous state equation of the linear steady system be:

$$x(t) = Ax(t) \tag{5}$$

In Eq. (5), the initial matrix **A** of the system is a companion matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \cdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{\mathrm{n}-1} \end{bmatrix} \tag{6}$$

The initial conditions are:

$$x\left(0^+\right)^T = \left[x_1\left(0^+\right)x_2\left(0^+\right)\cdots x_n\left(0^+\right)\right] \tag{7}$$

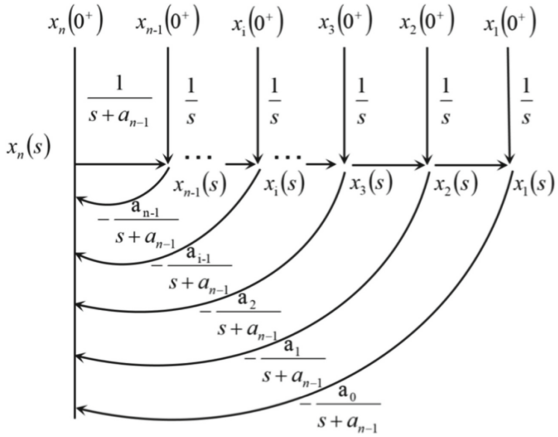

Fig. 2. Signal-flow graph

Perform Laplace transform on both sides of the equation and introduce them into initial conditions, then we get:

$$
\begin{aligned}
x_1(s) &= \frac{1}{s}x_1(0^+) + \frac{1}{s}x_2(s) \\
x_2(s) &= \frac{1}{s}x_2(0^+) + \frac{1}{s}x_3(s) \\
&\vdots \\
x_{n-1}(s) &= \frac{1}{s}x_{n-1}(0^+) + \frac{1}{s}x_n(s) \\
x_n(s) &= \frac{1}{s+a_{n-1}}x_n(0^+) - \frac{1}{s+a_{n-1}}[a_0x_1(s) \\
&\quad + a_1x_2(s) + \ldots + a_{n-2}x_{n-1}(s)]
\end{aligned} \tag{8}
$$

The corresponding signal-flow graph is shown in Fig. 2. Note that the input node refers to the initial state $\mathbf{x(0^+)}$ and the output node is the Laplace transforms of the state variable $\mathbf{x(s)}$.

From Fig. 2, it can be seen that due to the existence of $(\mathbf{n-1})^{\text{th}}$ touching loops $\mathbf{Li}$ ($\mathbf{i = 1}$, 2,……, $\mathbf{n-1}$), the characteristic equation of the system flow chart is Eq. (9).

$$
\Delta = 1 - \sum_{\mathrm{i}-1}^{\mathrm{n}-1} \mathrm{L_i} = \frac{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0}{s^{n-1}(s + a_{n-1})} \tag{9}
$$

In Eq. (9), $\sum_{i-1}^{n-1} L_i$ is the sum of various loop gains.

As can be seen in Fig. 2, there is only one forward path between $\mathbf{x(0^+)}$ and $\mathbf{x_i(s)}$ ($\mathbf{i = 1}$, **2**,……, **n**), and the path connects all loops. Thus, the input state matrix is calculated

as

$$\phi_{in}(s) = \frac{s^{i-1}}{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0} \tag{10}$$

There are two calculation cases when inputting the state matrix $\phi_{ij}(s)$, , where (**i = 1**, 2,…, **n**; **j = 1**, **2**,…, **n–1**). When **j** is greater than or equal to **i**, $\mathbf{x_i(0^+)}$ has only one forward path for $\mathbf{x_i(s)}$. The path touches $\mathbf{L_j, L_{j-1},\ldots, L_1}$ but not $\mathbf{L_{j+1}, L_{j+2},\ldots, L_{n-1}}$. This implies

$$\phi_{ij}(s) = \frac{s^{i-1}\left(s^{n-1} + a_{n-1}s^{n-j-1} + a_{n-2}s^{n-j-2} + \cdots + a_j\right)}{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0} \tag{11}$$

When **i** is greater than **j** and there exist **j**-th forward paths from $\mathbf{x_i\ (0^+)}$ to $\mathbf{x_i\ (s)}$, each of which is in contact with loops. This implies

$$\phi_{ij}(s) = -\frac{s^{i-j-1}\left(a_{j-1}s^{j-1} + a_{j-2}s^{j-2} + \cdots + a_1 s + a_0\right)}{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0} \tag{12}$$

Combining Eqs. (9) to (12), we can obtain Eq. (13) for solving the state-transition matrix.

$$\phi(s) = \frac{1}{s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0} \times \left[\phi_{ij}(s) \vdots \phi_{in}(s)\right] \tag{13}$$

In Eq. (13)

$$\phi_{in}(s) = s^{i-1} \tag{14}$$

If $\mathbf{j \geq i}$, then the state matrix $\phi_{ij}(s)$ is

$$\phi_{ij}(s) = s^{i-1}\left(s^{n-j} + a_{n-1}s^{n-j-1} + a_{n-2}s^{n-j-2} + \cdots + a_i\right) \tag{15}$$

If $\mathbf{i > j}$, then the state matrix $\phi_{ij}(s)$ is

$$\phi_{ij}(s) = -s^{i-j-1}\left(a_{j-1}s^{j-1} + a_{j-2}s^{j-2} + \cdots + a_1 s + a_0\right) \tag{16}$$

In Eqs. (15) and (16), **i = 1**, **2**,…, **n**; **j = 1**, **2**,…, **n–1**. From Eq. (13), Eq. (14) and Eq. (15), it can be seen that the state matrix $\phi_{in}(s)$, $\phi_{ij}(s)$ and $\phi(s)$ can be obtained only according to the values of $\mathbf{a_0, a_1,\ldots, a_{n-1}}$ in initial matrix **A** and the relative values of **i** and **j**. The state-transition matrix can be obtained by an inverse Laplace transform, i.e., $\phi(t) = L^{-1}[\phi(S)]$.

3.2 Solutions of State-Transition Matrix

Before the solution of state-transition matrix, what needs to be considered is to compress music sample data without causing information loss. When processing the data, time (seconds) and duration (seconds) can be fitted via Logistic regression. Figures 3 and

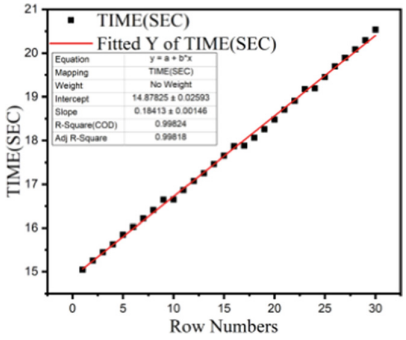

Fig. 3. Time fitting (seconds)

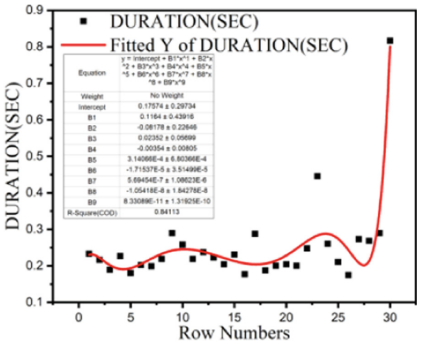

Fig. 4. Duration fitting (seconds)

4 shows that the sum of the squared errors for time and duration are 99.8% and 84%, respectively. We also obtain the output equations for time and duration via Figs. 3 and 4. Define the output equation for time as **y1**, and for duration **y2**. Since both data, pitch and note velocity, are distributed randomly, the fit is not effective.

For pitch data processing, lossless compression is achieved by converting decimal to octal and dividing by the same factor. When the original data needs to be recovered, we can multiply the factor and then convert octal back to decimal. Thus, the decimal data of the pitch is first converted into octal data, and then these octal numbers are connected at the beginning and end in three bytes. If there are not enough three bytes, then add a 0 to the forefront, and finally compress the whole pitch data into one decimal number. Since the converted number is large, and the multiplication and division operation are reversible, the initial data cannot be changed during recovery process. Therefore, we would normalize the number, that is, dividing the number by itself. Define the compressed value of pitch as parameter **a**. Similarly, divide the velocity value by its converted number to get a value 1. Define the compressed value of velocity as parameter **b**. This method can also be applied to equations. For instance, any of the polynomial equations being multiplied or divided by a number will only change the magnitude of final outputs, but not the distribution of the solutions of initial equation. During the music data pre-processing,

it is also necessary to understand the distribution of each coefficient, rather than relative values. For the equation $\mathbf{y_1}$ fitted in time (seconds), the corresponding output value can be solved directly by introducing time parameter **t**, since it is a very good fit and is a basic equation. Since the amplification or reduction transformation process is reversible and does not change its distribution, the output equation $\mathbf{y_2}$ is divided by itself and it becomes 1. To construct a stable distribution for Markov chains, the final equation of companion matrix should be designed as follows:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & -5 & 4 \end{bmatrix} \tag{17}$$

According to Eq. (17), combined with what has been mentioned above, we can obtain: $a_0 = -2$, $a_1 = 5$, and $a_2 = -4$.

According to Eq. (14), we can write that $\phi_{13}(s) = 1$, $\phi_{23}(s) = s$, and $\phi_{33}(s) = s^2$.

According to Eqs. (15) and (16), we can obtain

$$\phi_{11}(s) = s^2 + a_2 s + a_1, \phi_{12}(s) = s + a_2, \phi_{21}(s) = -a_0,$$
$$\phi_{22}(s) = s^2 + a_2 s, \phi_{31}(s) = -a_0 s, \phi_{32}(s) = -a_1 s - a_0.$$

Combining Eq. (13), we can obtain Eq. (18).

$$\phi(s) = \frac{1}{s^3 - 4s^2 + 5s - 1} \times \begin{bmatrix} s^2 - 4s + 5 & s - 4 & 1 \\ 2 & s^2 - 4s & s \\ 2s & -5s + 2 & s^2 \end{bmatrix} \tag{18}$$

Therefore, the final state-transition matrix can be defined as

$$\phi(t) = L^{-1}[\phi(s)] = \begin{bmatrix} e^{2t} - 2te^t & -2e^{2t} + 3te^t + 2e^t & e^{2t} - te^t - e^t \\ 2e^{2t} - 2te^t - 2e^t & -4e^{2t} + 3te^t + 5e^t & 2e^{2t} - te^t - 2e^t \\ 4e^{2t} - 2te^t - 4e^t & -8e^{2t} + 3te^t + 8e^t & 4e^{2t} - te^t - 3e^t \end{bmatrix} \tag{19}$$

3.3 Solutions of Acceptance Rate

In Eq. (19), when time $\mathbf{t} = 1$, then we can obtain

$$\phi(1) = \begin{bmatrix} 1.952 & -1.187 & 1.952 \\ 3.905 & -7.810 & 6.623 \\ 13.246 & -29.211 & 18.683 \end{bmatrix} \tag{20}$$

Next, we need to construct a stable distribution matrix and process Eq. (20), so that the sum of all numbers in each row of the matrix is one and do not contain negative numbers. Since the sum of each row is close to 2.7, the value of each row is converted according to 2.7. Thus, the state-transition matrix is obtained after the transformation when time $\mathbf{t} = 1$.

$$\phi'(1) = \begin{bmatrix} 0.345 & 0.310 & 0.345 \\ 0.366 & 0.239 & 0.395 \\ 0.466 & 0.009 & 0.525 \end{bmatrix} \tag{21}$$

Meanwhile, no matter what the initial state is, as long as the state-transition matrix is constant, the final acceptance rate will converge to a fixed value when the number of transfers **n** is large enough, so the initial acceptance rates are assumed to be 0.1, 0.2 and 0.7. Equation (21) and the initial reception rate are input into Markov chain Monte Carlo (MCMC) methods for calculation. After **n**-th transfers, the final convergence of the system is obtained, as shown in Eq. (22).

$$\alpha_N = (0.400068512, 0.16831977, 0.43099511) \tag{22}$$

4 Output Results

In Part 3, section B, the state-transition matrix of the sample data is obtained, and in section C, the acceptance rate of the sample data is solved. In this part, we can obtain the output matrix $A_1 = \alpha_N * \phi(1)$ for the sample data at time $\mathbf{t} = 1$, according to the MCMC method:

$$A_1 = \begin{pmatrix} 0.4005 & 0.1681 & 0.4308 \end{pmatrix} \tag{23}$$

Finally, the processed data is restored to its original state and decompressed, and then converted to MIDI format. The five-line staff of the music file is shown in Fig. 5.

In order to compare the experimental outputs, the same sample data is used here to substitute into the LSTM system for simulation, and the outputs obtained are shown in Fig. 6.

From Fig. 5 and Fig. 6, we can see that the outputs based on Monte Carlo method are better than those based on LSTM in terms of note distribution, as shown proportion of discordant notes in the generated music samples is smaller, and the difference of scale between notes is also within a reasonable range. In general, the distribution and articulation of notes appear to be more harmonious and natural compared with those

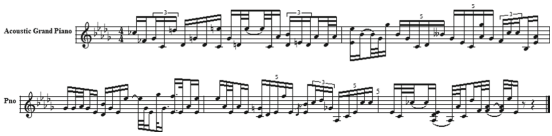

Fig. 5. Outputs of MCMC

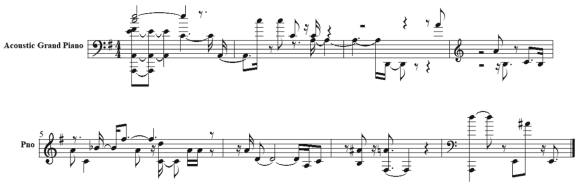

Fig. 6. Outputs of LSTM

generated by the LSTM method. However, by comparison, it can be seen that the note distribution of the music samples generated based on the Monte Carlo method is relatively compact, which is manifested in the auditory sense that the melody sounds relatively sharp. This is because the Monte Carlo method does not handle the time series of music well, which can result in poor fit. Thus, time series were extracted for separate processing.

5 Conclusion

This paper proposes an analysis framework for music generation based on Monte Carlo algorithm, which can be used to analyze the duration, pitch and velocity of notes in music samples, while for the time series of music samples, we adopted Logistic regression. The experimental results show that this method is an effective way to generate music. However, there are shortcomings and improvements in the method. For example, when dealing with music samples, it is important to consider how to better match the pitch and velocity with the time series. Furthermore, when there are too many different types of samples blended with the original music samples, Logistic regression is unable to fit the style of the original samples. Therefore, the process of identifying the style of music samples still requires manual manipulation or elimination of diverse samples before they are output. In conclusion, there is still much work to be done in the application of algorithm-based music generation, and this will be an ongoing research area in the future.

Acknowledgements. This work was funded by Science and Technology Winter Olympic Project (Grant No.: 2018YFF0300505).

References

1. Böck, S., F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer. 2016. MADMOM: A new python audio and music signal processing library. In *ACM International Conference on Multimedia*, pp 1174–1178.
2. Cho, K., B. Merrienboer, C. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoderdecoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pp 1724–1734.
3. Choi, K., G. Fazekas, K. Cho, and M. Sandler. 2018. The Effects of noisy labels on deep convolutional neural networks for music tagging. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2: 139–149.

4. Dorfer, M., J. Schlüter, A. Vall, F. Korzeniowski, and G. Widmer. 2018. End-to-end cross-modality retrieval with CCA projections and pairwise ranking loss. *International Journal of Multimedia Information Retrieval* 7: 117–128.
5. Joder, C., S. Essid, and G. Richard. 2011. A conditional random field framework for robust and scalable audio-toscore matching. *IEEE Transactions on Audio, Speech, and Language Processing* 19: 2385–2397.
6. Lu, K., C.S. Foo, K.K. Teh, H.D. Tran, and V.R. Chandrasekhar. 2019. Semi-supervised audio classification with consistency-based regularization. . In *Proceedings of the Interspeech*, pp 3654–3658.
7. Müller, M., A. Arzt, S. Balke, M. Dorfer, and G. Widmer. 2019. Cross-modal music retrieval and applications: An overview of key methodologies. *IEEE Signal Processing Magazine* 36: 52–62.
8. Paulus, J., M. Müller, and A. Klapuri. 2010. State of the art report: Audio-based music structure analysis. In *International Society for Music Information Retrieval Conference*, pp 625–636.
9. Pelchat, N., and C. Gelowitz. 2020. Neural network music genre classification. *Canadian Journal of Electrical and Computer Engineering* 43: 170–173.
10. Sangkloy, P., J. Lu, C. Fang, F. Yu, and J. Hays. 2017. Scribbler: Controlling deep image synthesis with sketch and color. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp 6836–6845.
11. Serrà, J., M. Müller, P. Grosche, and J. Arcos. 2014. Unsupervised music structure annotation by time series structure features and segment similarity. *IEEE Transactions on Multimedia* 16: 1229–1240.
12. Takamori, H., T. Nakatsuka, S. Fukayama, M. Goto, and S. Morishima. 2019. Audio-based automatic generation of a piano reduction score by considering the musical structure. In *International Conference on Multimedia Modeling*, pp 169–181.
13. Xu, M., Z. Wang, and G. Xia. 2019. Transferring piano performance control across environments. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp 221–225.
14. Yang, L.C., and A. Lerch. 2018. On the evaluation of generative models in music. *Neural Computing and Applications* 32: 4773–4784.