Musical patterns for prediction

Bachelor Project

Sebastian VELEZ DE VILLA

Supervised by M. Andrew MCLEOD

EPFL

Autumn semester 2020

Abstract

Repetitions play a central role in music, whether they be repeated themes, harmonies or even rhythms. They can differ in many ways: vertically, by shifting the notes up or down, time-dilated, by slowing or accelerating them, or just slightly different. We, humans, can easily recognise those patterns, but this task becomes incredibly difficult to implement algorithmically, as music is, by default, subjective, and thus non deterministic, as its perception can vary from an individual to the other. This paper aims at describing two ways of finding such repetitions, and how to use them to generate a continuation of a piece, given the first few notes. For discovery, a string-based and a translation-based approaches were used. Markov chains, where the next state depends only on the previous state, were employed for generation. My research aims at finding a solution to the first MIREX 2019 subtask, and at improving the already existing Markov chains approach. The results show a great improvement with both string-based and translation-based solutions, with the latter showcasing the most promising results. However, even if the generated notes are closer to the true continuations than the baseline, it still lacks some creativity, as the model only creates patterns or notes that already exist in the prime.

Contents

1 Introduction

My project focuses on the first MIREX 2019: Patterns for Prediction subtask, whose goal is to develop algorithms that take a short excerpt (called the prime) of music and produce a continuation: some notes that will follow the prime.

The main challenge for this task was to develop a good way to find patterns in music. When I say "good", I mean an algorithm that could find repeated sequences that would be considered as very similar by human listeners: it could be a repeated theme that goes up, or one played slower, or with a different rhythm. This trivial task for humans can become very cumbersome to implement for a computer, but not impossible. A basic way to find perfect matches would be to compare all subsets of the short excerpt with the whole set, and count how many times each one of them appears in the prime. This solution definitely works, but is slow, and doesn't work when the patterns are vertically shifted, or time-dilated. It is often the case that themes, or important part of the music, are repeated, but usually those themes slightly change throughout the piece. In other words, finding perfect matches is feasible, but does not find all the patterns.

The algorithms I want to implement are not new, however they were not designed specifically for this task, but for others such as music generation or music analysis.

Why is my research needed? Because repetition is a central key of music. We have all already had a song stuck in our heads, which definitely contained some form of repeated themes. Without them, music would be dull, uninteresting. It is therefore crucial to find those patterns, so that the music feels more natural for a human listener.

I know multiple models already exists, and a lot of AIs specifically designed for generation. I decided to work with statistics, and more specifically, Markov chains, because I wanted to learn something new, and different from what we can see nowadays. I also wanted to prove that a simpler model can still perform well, and produce convincing continuations.

In order to find such patterns, I decided to follow two approaches: a string-based approach, which finds exact patterns, and a translation-based approach, which finds exact patterns, as well as vertically shifted patterns. Consider the sequence "1 2 3 4 2 3 4 5" consisting of eight states. The string-based solution would only find the pattern "2 3 4", whereas the translation-based algorithm would find the pattern "1 2 3 4", knowing that it is shifted up by one to obtain "2 3 4 5".

As we will see below, those two approaches perform pretty well against other models, but still lack some creativity. I decided to only work with monophonic music, but the algorithms described below could be extended to polyphonic music.

2 Related work

2.1 AI approach

Various implementations of AI for music generation have been developed through the years. One of them, the Music Transformer ([1]) (built for a different task!) could be promising. This proposed architecture is based on the Transformer ([4]), which relies heavily on the concept of attention. The latter was first proposed by D. Bahdanau, K. Cho and Y. Bengio in 2014 ([3]), and by M.-T. Luong, H. Pham and C. D. Manning in 2015 ([5]). Attention models, as opposed to sequence-to-sequence models, pass down all hidden states from the encoder to the decoder, and give a certain score to each hidden state, so that the decoder focuses more on certain parts of the input sequence. This approach was first used for translation, where not all words in a sentence have the same importance, and was afterwards adapted for music generation. The goal is to produce music with long-term structure.

2.2 Statistical approach

A lot of my work was inspired by T. Collins, R. Laney, A. Willis and P. H. Garthwaite's work ([12]). In this paper, they discuss how a computer can generate music using statistics. The basic idea uses Markov chains, i.e., a succession of states, where each state only depends on the previous one.

A very simple music generation algorithm can be extended from that idea: generate the next note given only the previous note. Here's an example:

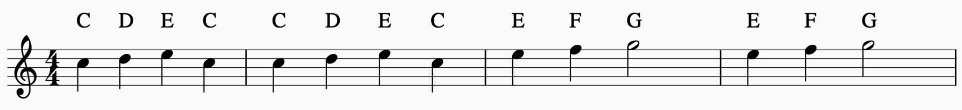


Figure 1: Simple melody

To calculate the probabilities for each state, a possible algorithm could be formulated in the following way: create a transition matrix, and for each note, count how many times it is being followed by all the notes, and divide by the number of times that note appears (disregarding the last note which, by definition, doesn't have a next state). As a result, this would generate the following table:

Pitch class	C	D	E	F	G
C	0.25	0.5	0.25	0	0
D	0	0	1.0	0	0
E	0.5	0	0	0.5	0
F	0	0	0	0	1.0
G	0	0	1.0	0	0

Table 1: Transition matrix of Figure 1

The previous transition matrix should be read line then column: "D is followed by E with probability 1.0", for example. This table can then be used to generate the next notes, by taking the last state, here G, and just reading the table with the probabilities for the next note. Suppose we want to generate 4 notes, a possible output would be: "E C E F". This simple generation algorithm can thus only generate states that appear in the prime sequence (i.e. the input), and only transitions that already exist (we can never have a C followed by a G, and we can never have a B, for example). There is therefore no notion of creativity or musical structure in this case, only repetitions of what already exists. To a human listener, most of the outputs from this algorithm do not sound good. However, it is fast, and can run in $\mathcal{O}(n + k)$, where n is the length of the sequence, and k is the number of generated states.

In the example I provided, a first-order Markov model was used to generate pitches, but it can also be extended for duration, and pairs of pitches and duration.

In order to improve this basic model, they propose a solution based on pattern discovery and pattern inheritance. A pattern (or motive) is basically a set of notes which is repeated, shifted, or time-dilated in the piece. In Figure 1, a pattern would be "C D E C", as it is repeated twice.

3 Design and implementation

Based on the pattern idea discussed in [12], I wanted to test two different ways of finding such patterns in monophonic pieces, and compare the results with the simple case algorithm described in subsection 2.2:

1. Exact non-overlapping non-shifted patterns (String-based).
2. Non-overlapping (possibly-) shifted patterns (Translation-based).

3.1 String-based pattern recognition

In the former case, the program uses a string-based approach, and finds exact patterns, which is defined as a sequence of notes that have the same pitches and duration, that appears at least twice in the score. A pattern detected by this algorithm for Figure 1 would be the first four notes. This method can obtain results in $\mathcal{O}(n^2)$.

3.2 Translation-based pattern recognition

In the latter case, the program uses translation vectors: it tries to find the maximum translatable pattern, i.e., a pattern that can be shifted either vertically or horizontally in the score. This algorithm is based on **SIA**, described in [11].

To make things simple, I will go step by step over this algorithm with the simple case scenario of Figure 1. For each note, calculate its translation vector with all the other future notes in the sequence: let n and m be two notes, with onsets and pitches, then the translation vector would be a 2-d vector, equal to $\begin{pmatrix} m.onset - n.onset \\ m.pitch - n.pitch \end{pmatrix}$. This would result in the following table for the first two measures, where the table should be read starting with columns and then rows (e.g. note $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$ can be transformed into $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$ by adding the vector $\begin{pmatrix} 0.25 \\ 2 \end{pmatrix}$:

(Onset, pitch)	(0.0, 72)	(0.25, 74)	(0.5, 76)	(0.75,72)	(1.0, 72)	(1.25, 74)	(1.5, 76)	(1.75, 72)
(0.0, 72)	-	-	-	-	-	-	-	-
(0.25, 74)	(0.25, 2)	-	-	-	-	-	-	-
(0.5, 76)	(0.5, 4)	(0.25, 2)	-	-	-	-	-	-
(0.75, 72)	(0.75, 0)	(0.5, -2)	(0.25, -4)	-	-	-	-	-
(1.0, 72)	(1.0, 0)	(0.75, -2)	(0.5, -4)	(0.25, 0)	-	-	-	-
(1.25, 74)	(1.25, 2)	(1.0, 0)	(0.75, -2)	(0.5, 2)	(0.25, 2)	-	-	-
(1.5, 76)	(1.5, 4)	(1.25, 2)	(1.0, 0)	(0.75, 4)	(0.5, 4)	(0.25, 2)	-	-
(1.75, 72)	(1.75, 0)	(1.5, -2)	(1.25, -4)	(1.0, 0)	(0.75, 0)	(0.5, -2)	(0.25, -4)	-

Table 2: Translation vectors of the first two measures of Figure 1, uncoloured non-empty cells are unique.

Then for each non-unique translation vector, add the head of the column to a sequence of notes, and sort them by length resulting in:

- $\begin{pmatrix} 0.25 \\ 2 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$, $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$, $\begin{pmatrix} 1.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 1.25 \\ 74 \end{pmatrix}$.
- $\begin{pmatrix} 1.0 \\ 0 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$, $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$, $\begin{pmatrix} 0.5 \\ 76 \end{pmatrix}$ and $\begin{pmatrix} 0.75 \\ 72 \end{pmatrix}$.
- $\begin{pmatrix} 0.5 \\ 4 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 1.0 \\ 72 \end{pmatrix}$.
- ...
- $\begin{pmatrix} 1.25 \\ 2 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$.

Then the patterns are filtered to keep only those which are contiguous, i.e., do not have a note from another pattern in between every pair of notes, like the first one in the enumeration above.

So as an example, the first four notes can be shifted by four beats to obtain the last four notes. This whole process can take $\mathcal{O}(n^2)$, but is slower than the string-based approach.

Once the patterns are found, both algorithms work in the same way, they transform the sequence of notes into a sequence of patterns, and apply a first-order Markov model on this transformed sequence to generate, not the next notes, but the next patterns. Then, the program translates back this sequence of patterns into a sequence of notes.

However, these algorithms only produce transitions that already exist in the prime sequence. To solve this problem, I decided to use some sort of additive smoothing. First, produce the transition matrix, as in Table 1. Then, multiply each probability by some number α (I decided to set $\alpha = 0.9$), and then distribute $1 - \alpha$ among all the states, according to their normalised histogram. For example, the normalised histogram for Figure 1 would be (with values rounded to the fourth decimal):

C	D	E	F	G
0.2857	0.1429	0.2857	0.1429	0.1429

Table 3: Normalised histogram of Figure 1

This approach would ensure that some states appear more than others, and would create transitions that don't exist in the prime, resulting in some "creativity". Taking this into account, this would result in the following transition matrix for Figure 1:

Pitch class	C	D	E	F	G
C	0.25357	0.46429	0.25357	0.01429	0.01429
D	0.02857	0.01429	0.92857	0.01429	0.01429
E	0.47857	0.01429	0.02857	0.46429	0.01429
F	0.02857	0.01429	0.02857	0.01429	0.91429
G	0.02857	0.01429	0.92857	0.01429	0.01429

Table 4: Transition matrix of Figure 1 with additive smoothing

The values in the table above have been rounded to the fifth decimal. The continuation would be produced in the same way as previously described, but taking these new values into account. This idea can also be applied (and was applied for the evaluation) in the two pattern recognition algorithms.

This was the last part of my work.

4 Evaluation

The evaluation of the continuation could be done manually, but these would take a lot of time, and would lack objectivity, as music is intrinsically differently perceived by everyone. Instead, an automatic evaluation was used, available at [9]. The dataset used for both generation and evaluation is composed of two parts, the prime, from which a continuation will be generated, and the true continuation, which will be compared with the outputs of my programs. The dataset was a subset of the Lakh MIDI dataset ([10]), which was specially prepared for this MIREX task.

The evaluation was done in two different ways ([9]): with a cardinality score, and with a pitch score.

4.1 Cardinality score

The cardinality score is defined as $\mathrm{CS}(\mathbf{P}, \mathbf{Q}) = \max_{t \in \mathbf{T}} \mid \{q \forall q \in \mathbf{Q} \mid (q + t) \in \mathbf{P}\} \mid$, where $\mathbf{P}$ and $\mathbf{Q}$ are two-dimensional sets of onset and pitch, $\mathbf{P}$ being the true continuation, $\mathbf{Q}$ being the algorithmic continuation, and $t \in \mathbf{T}$ being the vector that maximises the cardinality score. $\mathbf{T}$ is defined as $\mathbf{T} = \{p - q \forall p \in \mathbf{P}, q \in \mathbf{Q}\}$, and represents the set of all translation vectors that transform an algorithmically generated note into a note from the true continuation. From there, three methods of measurements can be computed:

1. The recall, which is the number of correctly predicted notes, divided by the length of the true continuation (we have -1 because at least one note can be translated into a note of the

true sequence, and we want the recall to be in $[0,1]$): Recall $= \frac{\mathrm{CS}(\mathbf{P},\mathbf{Q})-1}{|\mathbf{P}|-1}$. The higher the better.

2. The precision, which is the number of correctly predicted notes, divided by the length of the algorithmically generated continuation (same convention applies): Precision $= \frac{\mathrm{CS}(\mathbf{P},\mathbf{Q})-1}{|\mathbf{Q}|-1}$.

3. The F1 measure, which is basically a mix of the two previous methods.

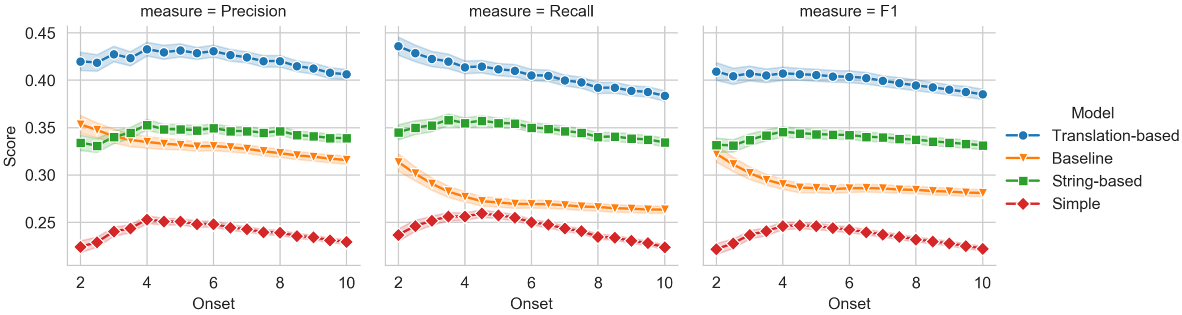


Figure 2: Cardinality scores for the three models described above and the baseline.

4.2 Pitch score

The problem with the cardinality score is that it is pitch invariant: if the generated continuation is equal to the true continuation but shifted vertically, it would get a perfect score (of 1), as we can translate the whole sequence of notes with only one translation vector. The pitch score is used to solve this. The pitch score is the comparison of the normalised histograms of the true and generated continuations, an example of a normalised histogram is given in Table 3. This process is repeated disregarding the octaves (mod 12 pitches).

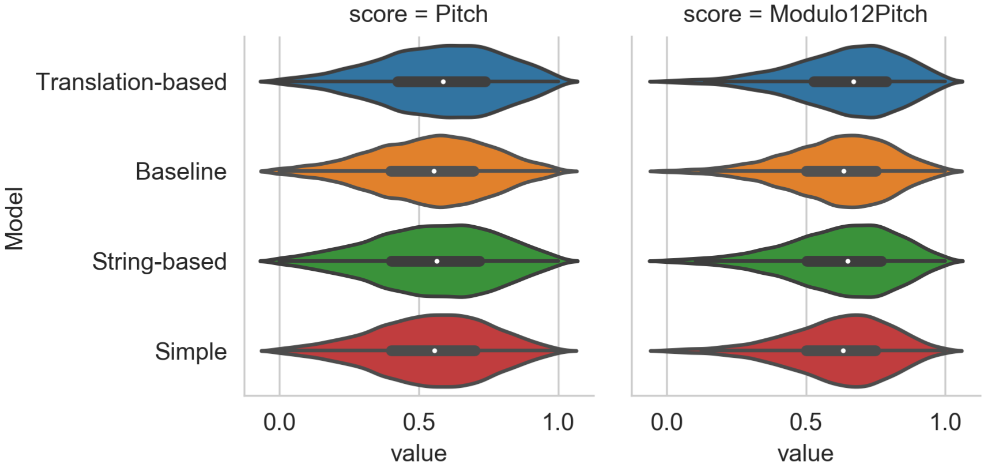


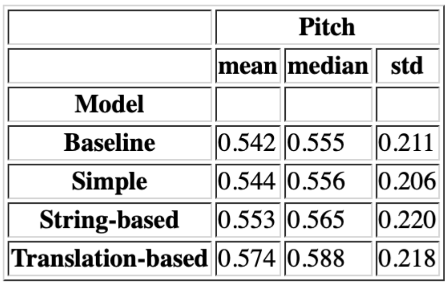
Figure 3: Pitch scores for the three models described above and the baseline.

4.3 Interpretation of the evaluation's results

The results shown in the pictures can vary from one execution to the other, as the generations are based on some randomness. However, the patterns discovered will always be deterministic.

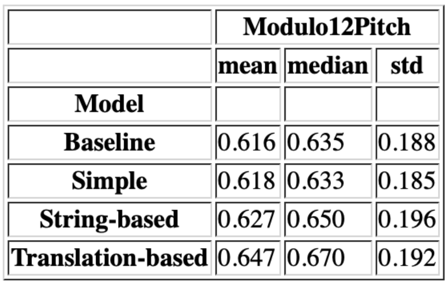
Starting with the cardinality scores, we observe that the simple first-order Markov model performs pretty badly, which can be expected. However, we notice a great improvement in the recall measure with the two pattern recognition algorithms (translation-based and string-based). This means that the continuation produced is closer in some sense, i.e., has a more similar shape, than the baseline, to the true continuation.

Concerning the pitch scores, all of them are pretty similar:



	Pitch		
	mean	median	std
Model			
Baseline	0.542	0.555	0.211
Simple	0.544	0.556	0.206
String-based	0.553	0.565	0.220
Translation-based	0.574	0.588	0.218

Figure 4: Pitch scores.



	Modulo12Pitch		
	mean	median	std
Model			
Baseline	0.616	0.635	0.188
Simple	0.618	0.633	0.185
String-based	0.627	0.650	0.196
Translation-based	0.647	0.670	0.192

Figure 5: Mod12 pitch scores.

We notice however that the translation-based continuation is again, better.

4.3.1 Comparison with an example

In order to compare the results, let's take the following piece, and see what the three algorithms produce as a continuation, as well as their corresponding cardinality and pitch scores:

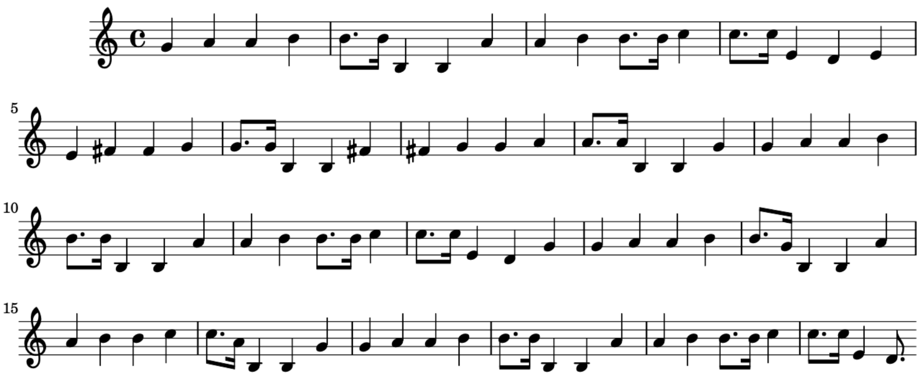
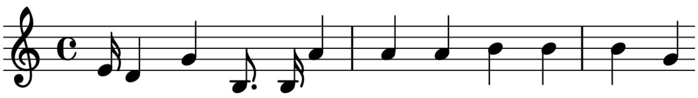
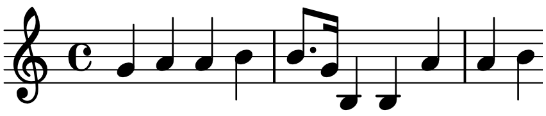


Figure 6: MIDI sample taken from the MIREX 2019: Patterns for prediction dataset [9].

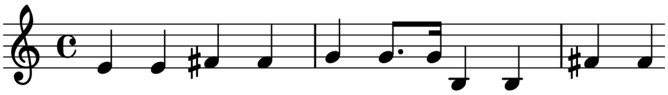
- Simple algorithm without pattern recognition:



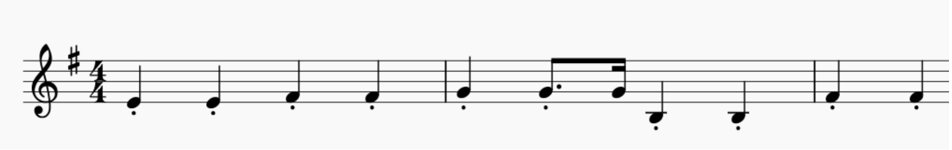
- String-based pattern recognition algorithm:



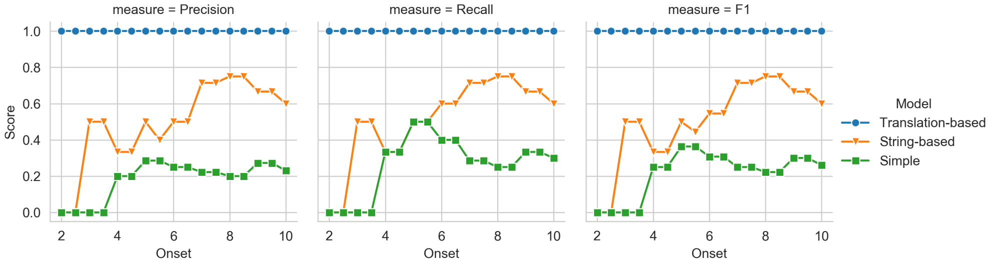
- Translation-based pattern recognition algorithm:



We can observe that in the last two cases, some patterns from the prime are repeated, whereas it is not the case for the simple algorithm without pattern recognition. Here is the true continuation of Figure 6:



Which gives us the following cardinality scores:



As shown in the graph, the translation-based approach obtains a perfect score for this prime, meaning that the generated continuation is equal to the true continuation. The string-based algorithm performs slightly better than the simple model. Concerning the pitch scores, we obtain the following means: **0.143** for the simple algorithm, **0.621** for string-based and **1.000** for translation-based. Thus, if we have both a higher cardinality and pitch scores, then we can consider the continuation as better. Note that these values might change between different execution, as the results depends on some randomness.

The produced outputs from the best version, the translation-based approach, heavily depend on the structure of the prime. It has to be repetitive, i.e. contain patterns (either vertically or horizontally shifted, but not time dilated), and long enough to detect multiple patterns, to add some variety. If the prime doesn't respect one of these conditions, then the generated continuation will be poor.

5 Conclusion

Overall, the two pattern-based algorithms work well when the true continuation contains a repetition of a known pattern, i.e., a pattern that has already appeared at least twice in the prime. It performs badly when the continuation is new, as the algorithms do not have a notion of creativity. Although these algorithms perform pretty well as we saw in section 4, further work can be done. Each solution I provided was an improvement of the previous version, and improvements over the last version can still be made, by adding non-exact patterns to the translation vectors approach. For example, the two patterns "C D E C" and "C D C C" could be considered as similar, as most of the notes are equal. Another approach would be to use a second-order Markov model, i.e., where the next state depends on the two previous states.

Similar algorithms for polyphonic music can be developed, as well as AI based solutions, which could be combined with statistical models. Overall, this statistical approach is great for repetitive pieces, but certainly lacks creativity, which remains a central piece of composition.

6 Bibliography

References

[1] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, I., Simon, C. Hawthorne, Andrew M. Dai, M. Hoffman, M. Dinculescu and D. Eck. Music Transformer, arXiv: Learning, 2018. https://arxiv.org/abs/1809.04281

[2] Peter Shaw, Jakob Uszkoreit and Ashish Vaswani. Self-attention with Relative Position Representations, arXiv: 2018. https://arxiv.org/abs/1803.02155.

[3] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, arXiv: 2014. https://arxiv.org/abs/1409.0473.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. Attention is all you need, arXiv: 2017. https://arxiv.org/abs/1706.03762.

[5] Minh-Thang Luong, Hieu Pham and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation, arXiv: 2015. https://arxiv.org/abs/1508.04025.

[6] Oore, S., Simon, I., Dieleman, S. et al. This time with feeling: learning expressive musical performance. Neural Comput and Applic 32, 955–967 (2020). https://doi.org/10.1007/s00521-018-3758-9

[7] Ziyu Wang, Yiyi Zhang, Yixiao Zhang, Junyan Jiang, Ruihan Yang, Gus Xia and Junbo Zhao. PIANOTREE VAE: Structured Representation Learning for Polyphonic Music. https://program.ismir2020.net/poster_3-06.html

[8] Tom Collins's workshop: maia-suggest and maia-snake. https://glitch.com/@tomthecollins/wi-mir-2020-workshop

[9] MIREX dataset: https://www.music-ir.org/mirex/wiki/2019:Patterns_for_Prediction

[10] Lakh dataset: http://colinraffel.com/projects/lmd/

[11] David Meredith, Kjell Lemström and Geraint A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music, March 10, 2003. http://www.titanmusic.com/papers/public/cmpc2003.pdf

[12] Tom Collins, Robin Laney, Alistair Willis and Paul H. Garthwaite. Chopin, mazurkas and Markov: Making music in styles with statistics. December 2011. https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2011.00519.x

[13] Pierre-Yves Rolland (1999) Discovering Patterns in Musical Sequences, Journal of New Music Research, 28:4, 334-350. https://doi.org/10.1076/0929-8215(199912)28:04;1-O;FT334