Music Generation from MIDI datasets

Moritz Hilscher[1], Novin Shahroudi[2]

Institute of Computer Science, University of Tartu
[1]moritz.hilscher@student.hpi.de, [2]novin@ut.ee

Abstract. Many approaches are being used for music generation to this date. Many fields are revolutionized by the recent deep learning breakthroughs. Music generation is no exception. In this work we employ a character-based Recurrent Neural Network (RNN) for music generation. RNNs has renowned for modeling sequential data and has become popular in the deep learning community for many applications.

1 Introduction

Similar to many other fields such as speech recognition, text generation, image captioning, etc. music generation is hugely influenced by neural networks recently. Our aim in this project is to replicate one of the successful approaches and demonstrate our findings throughout this process.

Our approach is inspired by similar attempts by [2,1] which shows that Recurrent Neural Networks (RNN) with relatively straight forward architecture have the capacity to generate pieces of music. Musics are temporal data containing dynamics that makes them complicated for computer models to generate one. RNN has shown to be a promising method to predict sequential data especially text sequences. To this end, many attempts are made to date for generation of music using this method, or other methods such as CBOW, sequence-to-sequence, or GAN [1]. Among music representations such as MusicXML or the ABC notation, we use MIDI files which are widely available and convert them into a simple textual representation (which is also a benefit while working with RNNs). The textual representation of a MIDI music track is generated by quantizing played notes to time steps of a specific length and then specifying with a different character for each note which note is played at which time step.

In section 1, we give an overview on the music generation/composition field, and more specifically deep learning approaches that is being used to date for this task. Section 2 explains data preparation and character-RNN. Results in section 3.4 show that with fair amount of implementation efforts and dark arts our approach is good for music generation. Moreoever, further discussions, possible extensions of the work with conclusion are covered respectively in sections 4, 5, and 6.

2 Background

Music composition using computer aided systems has a long history since appearance of computers and digital systems. There are wide range of technologies utilized in different levels such music synthesizers, and improvisers. Alan Turing was the first to generate notes using computers in late 40s. Later in 1950s attempts resulted to musics that being played through computers but limited to a exact repertoire of the pieces of music using algorithmic composition which derived from mathematical descriptions. As music synthesizers became more prevalent, more standards emerged. In 1980s Japanese personal computers employed audio programming languages such as Music Macro Language (MML) and MIDI interfaces. At this stage musics could be generated in real-time.

In general computerized music composition can be categorized into **computer-generated musics**, and **machine improvisation**. There are different approaches and applications for **computer-generated musics**. Figure 1 shows relation between each of these approaches.

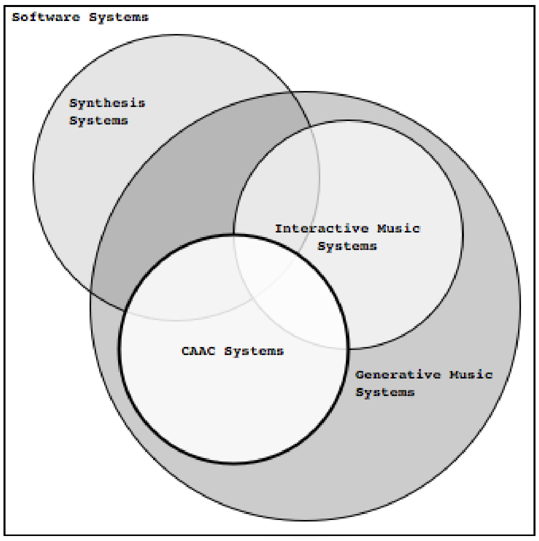

Fig. 1: Diagram of methods for computer-generated musics

Machine improvisation which is relatively a newer concept employs artificial intelligence and machine learning to capture characteristics of the music (so-called musical styles). Compared to algorithmic composition these methods rely on available examples to analyze and capture these styles. Furthermore machines are able to compose a music or make their own imitations based on previously seen samples.

Markov Chain and stochastic processes were the first approaches being used as a statistical modeling which enables capturing patterns. Modern methods are loss-less data compression, prefix suffix tree, and string searching.

Neural models also known as "deep learning" use music data to analyze and model content of the music to generate a new music. They are also called as data-driven approach. These models are essentially layered computational graphs that each deeper level contain more sophisticated yet higher level features derived from the input. Graph nodes (so-called neurons) receive input from other nodes and propagate output based on a weight and activation policy. Recurrent Neural Networks with LSTM cells are of more attention for music generation because of their appropriate structure which makes them suitable for sequential and recurring patterns that also exist in every piece of music.

In recent project report of N. Agarwala, et.al [1] they are able to generate monophonic and polyphonic music in different genres without any predefined music composition rules using character RNN[1], and Seq-to-seq[2] with GRU[3] and LSTM[4] including bi-directionality and attention. Their character based RNN learns music patterns with accuracy of 60% being able to fool 37% of the human listeners, yet the generated music is not structurally correct. Their sequence-to-sequence models reach accuracy of 65% with 75% success rate in fooling the listeners with corrected music structure. They also implemented GAN but according to their claim it does not produce meaningful results due to the instable training. To have a sensible measure of their character RNN they ran the network with the same dataset that character RNN presented by Kaparthy et.al [].

Their results show that increasing hidden layer size and shorter embedding size increases accuracy of the character RNN. They also showed that increasing both hidden layer and embedding size gives better accuracy for the seq-to-seq model. In addition, they employed ABC notation and claimed that notations such as MIDI are sound-based and may not be transcribed properly compared to ABC. In contrast, they try to come up with models that are able to learn the *grammar* of the music.

Similar attempt by Bob L. Sturm, et.al [2] uses character based and token based approach to train LSTM network. Albeit, their focus is on developing models that facilitate music composition. They have made statistical analysis of the generated musics that can help relating the synthesis to the original examples. They were able to use aforementioned approaches to generate structurally valid musics. Their approach can be a inspirational recipe for regularizing the RNN models to generate grammatically valid musics. One of their achievements was to employ RNN models with thousands of hidden nodes trained on thousands of training samples.

[1] Recurrent Neural Network
[2] Using Encoder decoder paradigm
[3] Gated Recurrent Unit
[4] Long-short term Memory

3 Methods

Here, we explain MIDI file format, how to generate text sequences from, and the normalization in the data preparation. Further, we explain our model and approaches to learn the patterns in the text sequence, and how to generate with in the network architecture.

3.1 Data Preparation

We use textual representation of a piece of music extracted from MIDI files to feed it to the network. Each MIDI file is a collection of tracks containing events. Tracks can be different instruments but also left and right hand for a piano music piece. Events represent note on/off, tempo, instrument-type, meta information about the music, and etc. Also each event contains a tick number and a payload. Tick number is the lowest level time resolution. The tick number of each event describes how many ticks passed from the previous event to the current one. Notion of time in the music is defined using the tick, tempo and resolution. Tempo is number of beats per minute (BPM) also referred to as Quarter note per minute (QPM). The resolution is the pulses per quarter-note (PPQ), thus describing how many ticks per quarter note being played pass. The higher the resolution the shorter in time is one tick. Fig. 2 depicts this explanation.

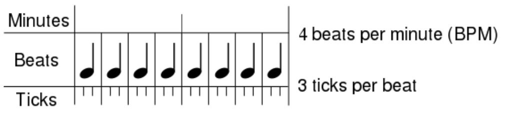

Fig. 2: Diagram of methods for computer-generated musics

Although a MIDI file can be comprised of different instruments/tracks, we treat events from different tracks as events from one track. Further, we quantize MIDI events by a specific divider of MIDI resolution. For example all events are quantized to 1/4 or 1/8 of MIDI resolution. We started with 8 as a quantization divider, but ended up using 4 instead (4 still yields reasonable musical quality compared to original pieces, 8 already yields quite long text with many repeated chords). Each of these 1/4 parts of a quarter note, hereafter, a time step is then used to extract notes of all events within that time step to construct the textual representation of the MIDI. Each note pitch is represented with an ASCII character. ASCII characters concatenated together represent notes being played at the same time step (also known as chord). Next and previous chords/time steps are separated using a space.

After some experiments, we noticed that MIDI tracks have quite different tempos. The length of generated MIDI music depends on the quantization (quantization divider) and on the note types (quarter, eight, etc. notes). Although these would make much of different in the way that the music is being heard by us but because of the way we represent the music in text it looks like the same but actually it is not. That brings the necessity to normalize the data so that every piece has at least almost the same tempo. To normalize, MIDI events are morphed with their tick timings to the same tempo. That arises the problem that the notes do not line up with quantization dividers (as notes are not like quarter, eigth notes anymore but other fractions). To minimize this misalignment, the tempo is doubled/halved, etc. until we are nearest to desired tempo (average tempo of dataset) and yet keep the misalignment low.

To overcome small size of the training set we also augmented the dataset by transposing all the tracks by one or more semitones (one pitch level) up/down. This might enable the network to learn harmonies as well as having more variety of notes in just more than one key available.

3.2 Network Architecture

We use supervised learning in order to train our model using a character-based RNN with LSTM cells with a many-to-one architecture. In order to formulate our problem into supervised learning, we use sentences from the dataset as input (x) and the character following this sentence in the dataset as label (y). Input sentences are substrings from the dataset (sentence length equal to the number of RNN time steps) generated by sliding a window of sentence size and specific step size over all the dataset texts. Moreover, our network inputs and outputs are hot-one-vectors since we have a fixed vocabulary of note characters (and the space to separate music time steps).

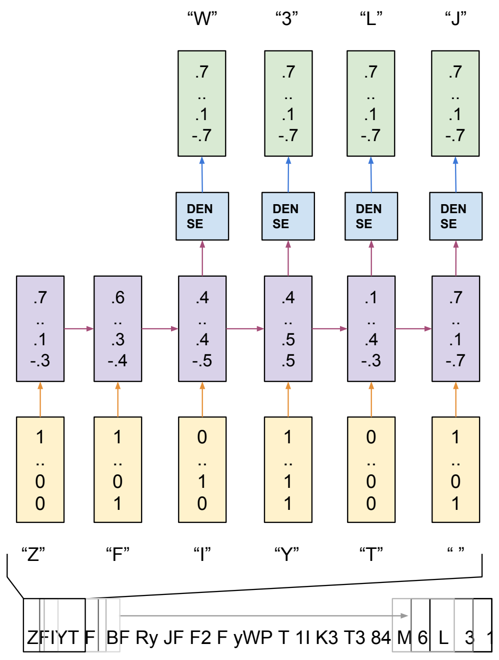

Fig. 3: character-based RNN architecture

As depicted in Fig. 3 we use a dense layer with softmax activation which in the end gives us a probability of a one-hot-vector. It essentially contains probability of each possible music text character. We also split the dataset into training and validation sets by randomly taking tracks until desired percentage (measured with length of tracks textual representations) of each is covered.

3.3 Sampling

After the network is trained with the textual representation of MIDI music, new music can be generated by generating text using the trained model which is then converted back to MIDI to playback. To generate new text we feed the network with a warm up sequence. This can either be a random character sequence (with dataset vocabulary), a sentence from the dataset, or even a sentence from a track not included in the dataset (but with same vocabulary). When the warm up sequence is fed into the network, the next character following the warm up sequence can be predicted. Newly predicted character(s) are appended to the end of the previous input as the input slides and maintain its fixed length for the input. As explained in the architecture part, the prediction is a vector of probabilities for each text character. Before we nominate a character as the network's prediction we divide it by a parameter named temperature. More temperatures lead to more uniform probability and temperatures closer to zero lead to probability distribution of the vector. The concept is very close to the simulated annealing algorithm.

```
U IQU JNS JNS JNZ JNZ zIX zIX zIQ zIQ zGV zGV zGP zGP EQ EQ IQ IQ LQ
Q LQ Q L L L7 L7 4 4 1 1 Q9 Q9 9Q 9Q 9O 7O 9Q 6Q 7 2 96 7 J% J% N% N%
%Q Q U X L L Q U LX LX UQ UX LX SV 7Q 9Q 7L 7L V6 7V 4P V4 U6 V4 2N
2N 1V V4 2N L2 2U 2U S1 S1 ZQ ZQ Q Q P Q 2N 2N 2Q 2Q 2N 2N 2Q Q 7N 7N
7Q 7Q L6 L6 9Q 9Q 7L 7L U6 U6 N4 N4 2U 2U K1 K1 2N 2N 2UQ 2UQ 2UQ 2UQ
S2 S2 L4X LX K5 K5 LV4 LV4 S4 S4 S2 S2 L4P L4P SL4 SL4 L2Q L2Q SL1
SL1 SKZ SKZ SL1 SL1 EN EN ZEN ZEN XLP XLP XLP XLP XLP XLP XLP XLP ENW
ENW ENW ENW UNG UGQ SNG SNG QXI LXI ZLI LXI SL1 SL1 JS2 JS2 L4 L2 L1
ZL 0S S2 0P 1P S4 S6 8P 8P L9 L9 PP ’S ’S ’P ’P S" S" L9 L9 PP L" L9
6I 6I N4 N4 9I 9I F6 F6 "I "I N" N" 9G 9G 8N 8N D6 D4 K4 N4 Z Z 4 4
DL1G DL1G ZDLG D1LG ZDLG DLGX DLWG DLGX Z 1 2 1 Z 1 6 4 2EI ZEI ZEI
2EI X Z X W X X X W UI SI Q S U S Q P Q Q Q U X 1 4 w6 9y " 8 7 4 1 Z
X W U SX XQ Z 1 2 1 2 1 ZN XP WQ U S U W X ZE 2N 2 S1 2N 4Q 5Q 8W S9
9X S9 9X 8N 8Q 8X 8W 8P 8S 9X S9 "Q 9Q 8 9Q ’P S
```

Fig. 4: A sample output of the sampling stage

3.4 Determine over-fitting

We used wave signals of the music to compare the generated music and training dataset to find correlation between them. First all musics turned into frequency domain using Fast Fourier Transform. Then, by performing autocorrelation and

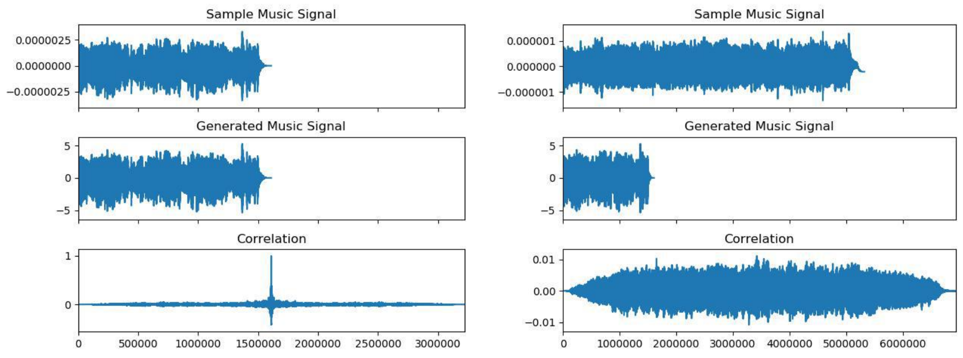

Fig. 5: Left figure shows two similar input signals with the expected correlation that reached to 1 on the 3^{rd} row. Right figure is a sample performed on a generated music and the training data.

measuring the maximum correlation points between two piece, we can have a rough measure of how much our generated music may be overfitted to the original pieces.

As the correlation is performed in frequency domain it makes it a versatile method for our purpose and does not depend on sequences. This saves us from hassle of textual pattern analysis which could be the approach for this purpose. Although the correlation process could be very time consuming on a regular processor it takes less than a minute to perform on our dataset size for one generated piece of music of about 30 seconds.

4 Experiments & Results

We implementation our model using Keras library and mainly trained on Google Cloud. We employed following data sets (scraped from different websites):

- Pieces by Bach (194 tracks, 1.4M characters text)
- Pieces by Mozart (39, 0.4M characters text)
- Pieces by different composers, pianomidide (335 tracks, 2.3M characters text / filtered: 132 tracks, 0.8M characters)

Different configurations of the discussed network tested and evaluated as you can find the details in table 1. Important practices that lead to the best results are accuracy evaluation using training/validation set, tempo normalization, data augmentation, batch data shuffling at each epoch. Other parameters of the network that were the same through out all the reported results are as following:

- Loss: categorical cross entropy
- Optimizer: Adam
- Learning rate: 0.001
- Validation split: 0.2

We found that we achieved the best musical result with a many-to-one RNN with one LSTM layer (512 units), sentence length 100, trained for the Mozart

Table 1: Results of some recently trained configurations

Model	Layers	LSTM Units	in_out_step[a]	Dataset	Data[*]	Epoch	Acc	Epochs	Acc
1	1 LSTM	100	100_1_1	bachsmallst	N,T,S	15	53%	30+	99%
2	1 LSTM	512	100_1_1	mozart	N,T,S	3	69%	13+	86%
3	1 LSTM	512	100_1_1	bachsubset	N,S	5	68%	23+	79%
4	2 LSTM	256	100_1_1	mozart	N,T,S	5	69%	36+	96%
5	2 LSTM	256	100_1_1	pianomidide	N,T	12	62%	17+	85%
6	1 LSTM	256	100_1_1	pianomidide	N,T	6	64%	39+	85%
7	2 LSTM	256	100_1_1	mozart	N,T	5	85%	24+	92%
8	2 LSTM	420	100_1_1	bach+mozart	N,T	3	72%	14+	85%
9	1 LSTM	256	100_1_3	mediumq4	None	10	60%	10+	58%
10	2 LSTM + 2 FC	256	100_1_3	mediumq4	None	10	60%	10+	58%
11	1 LSTM	256	100_5_10	mediumq4	None	18	45%	n/a	n/a
12	1 LSTM	512	100_5_10	mediumq4	None	18	46%	n/a	n/a
13	2 LSTM	256	100_5_10	mediumq4	None	30	43%	n/a	n/a

[*] N: Normalization, T: Transpose, S: Shuffling
[a] input length, output length, step length

dataset (normalized tempos, augmented data with all pitches transposed by six tone differences $[0, 5]$). Results sound quite harmonic compared to other trained networks, which we think is mainly due to the data augmentation. Unfortunately we were not able to train the bigger Bach dataset with this data augmentation as we ran into memory issues and had some time problems. In general, we discovered the following properties about the different parameters of our networks architecture:

– Sentence length = RNN time steps: With the used quantization divider of 4, 100 as sentence length seems like a good fit providing the network enough context about previously played chords.
– Many-to-many architecture (outputting n next sentence characters instead of just one), convolution layer: Didn't change the achieved accuracies a lot, and also didn't much improve the musical quality.
– Number of LSTM layers/units: We discovered from our training experiments and other text generation approaches that the number of network parameters (mainly achieved by changing number of LSTM layers/units) should about match the size of the dataset (number of characters) in magnitude. We figured out, though, that in this case we might want the network to overfit a bit to reproduce some of the harmonic sounding note combinations/chord progressions from the training tracks.
– Tempo normalization: In the beginning of the training experiments, we had some issues with the network producing tracks that had some chords repeating for a long times. Tempo normalization seemed to fix that issue.
– Data augmentation: Transposing notes to different keys in the training data has showed to be very good to improve the harmonies of generated tracks as shown with the best network trained on the Mozart dataset.

Based on our experiments the generated pieces by the network are not performing a fool immitation of the training data as one sample is being shown in

Fig. 6 there is a very small proportion of correlation between the training set pieces and the generated piece which is acceptable as long as the model is training patterns from each piece because there are always a chance for producing similar patterns as those the model trained on.

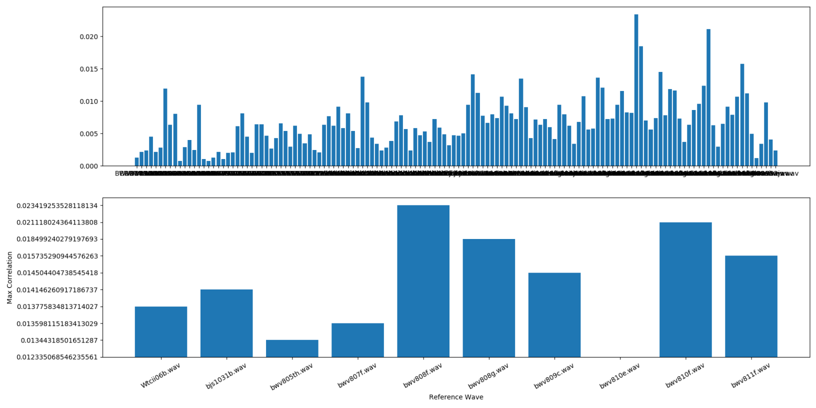

Fig. 6: Top plot shows correlation with the whole dataset, the bottom is the top-10 highly correlated ones

5 Discussion

Although the implemented approach given descent results it is bound to the limited knowledge of the music that we provide as the input. Many other characteristics of the music cannot be captured through the current textual representation. The pitch level which corresponds to the intensity or volume of echo note is one of those. From experiments with the temperature parameter for the sampling stage it is evident that network has not seen enough training data as for the lower temperatures it produces repeated notes/chords.

In the worst case scenario overfitting may occur and get repeated if the input sequences be as same as those the network being trained on. However, because of randomness nature of the sampling incorporated using temperature parameter the chances that the generated music would be a total imitation of the trained data is very low. Further analysis of the temperature effects on the generated music can be done for future.

6 Future works

Here we list possible future works of this work:

- Other RNN architectures and combination of layers
- Encoder-decoder architecture with sequence to sequence approach
- Experiment with different augmentation of different keys in order to improve musical harmonies of results
- Fully training of all current datasets combined with large data augmentation (was not possible yet to memory and computational power issues)
- Training using different datasets (additionally to mainly Bach and Mozart dataset currently in use)
- Training using different genres at the same time
- Other music notations
- Incorporating the velocity (not to be confused with tempo; MIDI velocity describes how loud a note is played)
- Multiple tracks/instruments
- Textual pattern matching algorithms to determine overfitting of the model and getting better insights at the generated texts

7 Conclusion

We implemented a character-based RNN using LSTM cells on classic piano pieces using MIDI files. The best model reached training accuracy above 90% on some of the datasets and about 70% validation accuracy. Although the model would generate repeating notes/chords for some reasons, our findings show that repetitions can be eliminated to some level that produces relatively satisfactory results by tempo normalization and data augmentation (transposing notes). Also the accuracy is highly depended on the aforementioned. Beside available samples for listening we used autocorrelation to have a measure of overfitting of the model. Some of the selected pieces of this work are available at `https://yellow-ray.de/~moritz/midi_rnn/examples.html`

8 Contributions

Both authors contributed to the brainstorming, network training, documentation, presentation and reports of the project. Moritz Hilscher contributed to the finding of datasets, data preparation, network model implementation. Novin Shahroudi contributed to the Jazz dataset (not reported here), network model, and autocorrelation implementation.

References

1. Yuki Inoue Nipun Agarwala and Alex Sly. Music composition using recurrent neural networks. Technical report, 2017.
2. Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *CoRR*, abs/1604.08723, 2016.