Piano Music Generation with LSTM

Jean-Baptiste Gourlet
CentraleSupélec, Université Paris-Saclay
Gif-Sur-Yvette, France
jean-baptiste.gourlet@student.ecp.fr

Grégoire Gambino
CentraleSupélec, Université Paris-Saclay
Gif-Sur-Yvette, France
gregoire.gambino@student.ecp.fr

Abstract

In this paper, we discuss a way of generating music artificially. What is used are LSTM Recurent Neural Networks. We focus on piano and use as a training dataset classic music played by professionals. A challenge of this project was the data representation. Choices had to be performed to best represent the sound a human can hear and enable the model to learn efficiently some patterns. The methodology of network hyperparameter choices is also discussed. Final results are promising and offer a relatively good sounding output.

1. Introduction

1.1. History

Algorithmic music generation is not a recent matter of interest. Indeed, Greek mathematicians felt that there was a relationship between the harmony of sounds and the laws of nature. Music and numbers were linked as musical song corresponds to a time series of numbers. One of the first algorithm to generate music was implemented by Mozart, it is known as *Musikalisches Würfelspiel* [4]. 16 pre-composed measures are randomly chosen with a dice [7].

An alternative to this stochastic approach is a rule-based approach [2]. Rules of structure, rhythm, repetition and others can be implemented to increase the probability of composing a good sounding sound.

Recent algorithmic approaches are based on Artificial Intelligence. Like the rule-based approach, this technique uses a grammar, but the algorithm is able to learn it from external examples. This approach is the one that we will focus on in this project.

The code repositery and some samples of inputs data can be found here: `https://github.com/jbgour/Music-Generation-Project`

2. Background

2.1. LSTM networks

Recurrent neural networks, and especially LSTMs, work like a musician would improvise with his instrument. It is able to remember a previous note sequence and predict – therefore play, the next sequence that is the most appropriate to the latter. Moreover, such RNNs are able to remember long-term dependencies and could be used to create a structured music track.

2.2. Music Representation

MIDI files are often used professionnaly to numerize music and exploit it.

MIDI is a communication protocol for digital musical instruments: a symbolic representation, transmitted serially, that indicates Note On and Note Off events and allows for a high temporal sampling rate. The loudness of each note in a discrete quantity referred to as velocity (the name is derived from how fast a piano key is pressed). While MIDI encodes note timing and duration, it does not encode qualities such as timbre; instead, MIDI events are used to trigger playback of audio samples. [7] In a first approach, we have decided to simplify the data representation and not consider the duration and the velocity of each note. Therefore, each note will have the same duration and be played one after another at a rate of 120BPms. Such simplification decomplexifies the model at the expense of not being able to reproduce a most usual way of playing the piano : playing chords in the left hand and a melody with the right hand over it.

In order to digitalize each pitch into a valid input for a model, we have identified two different techniques :

- A non-ordinal representation of the set of key by a 1x128 binary vector with a 1 for each key pressed at time t. This representation has the benefit of display some similarities between a single C note for instance, and the C chord with the single note as a base line.

- An ordinal representation by associating each piano key a number from 1 to 88. This representation uses the fact that each pitch is associated to a certain frequency – for instance A4 = 440hz, and therefore classify the notes in an ascending manner. However, for the chords, it is hard to tell how the chords and single notes should be ordered between them. In this paper, we chose to sort each chord alphabetically using their letter notation e.g C4.D5.A6 and to attribute them a number starting from 89.

We chose the ordinal representation - a future work would have to be done comparing both methods.

3. Approach

This section describes the models we implemented.

3.1. Data source

The data used in this project was taken from recordings of the *Minnesota International Piano-e-Competition*[1], an online piano competition. 700 hours of classic piano music was collected. This dataset has the following benefits [5] :

- All samples are from classical music
- Only one instrument is present in the recording: the model will have less difficulty to predict music while for a polyphonic recording, a much more complex training would be necessary.
- The artists are considered as experts one. Therefore, the quality of inputs provided to the neural network might not be a source of poor generation quality.

3.2. Data Representation

First thing to do is to transform the MIDI data into a format that the neural network will take as an input. We extracted the notes of the MIDI file. A note is as an example: 'C3'. We also extracted chords which are combination of notes. An example of chord could be 'B-3.F3.D3', which matches to three notes being pressed at the same time on the piano.

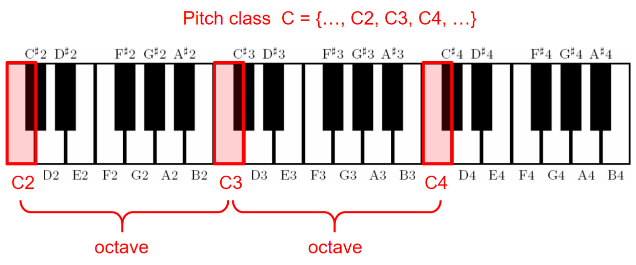

Pitch class C = {..., C2, C3, C4, ...}
C♯2 D♯2 F♯2 G♯2 A♯2 C♯3 D♯3 F♯3 G♯3 A♯3 C♯4 D♯4 F♯4 G♯4 A♯4
C2 D2 E2 F2 G2 A2 B2 C3 D3 E3 F3 G3 A3 B3 C4 D4 E4 F4 G4 A4 B4
octave
octave

Figure 1. Correspondance between piches and piano keys `www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S1_MusicalNotesPitches.html`

So as to be able to compute metrics, we transformed those pitches that are from type string to float. To do so, we sorted the pitches given the alphabetical order and attribute them an ascending float. That makes sense from a musical point of view. Indeed, notes that would be embedded closely are from the same note, but not necessarily the same frequency. All pitches of the vocabulary are transformed into a float. We normalize the inputs by dividing by the number of pitches so as to have the between 0 and 1.

Next step is to adapt the list of embedded pitches into a structure that will be used by the LSTM network. Goal of the network is to predict a pitch at time *t* given a list of pitches at time $t_{-sequence-length}$,... t_{-1}.

Hence, input to the the network is a list of list of 50 embedded pitches. Target is the the embedded pitch following this time series.

3.3. Model

For the architecture of the model, we have been using LSTMs layers complemented for 3 different type of layers. To make the model fully connected, we used Dense Layers. Moreover, Dropout layers have been used to create generalization – a regularization technique consisting of setting an input units to 0 at each update during the training to prevent overfitting. At the endpoint, the Activation layer is added to the model, which helps in deciding, which neurons (LSTM cells) should be activated and whether the information gained by the neuron is relevant, making activation function highly important in a deep neural network. [7]

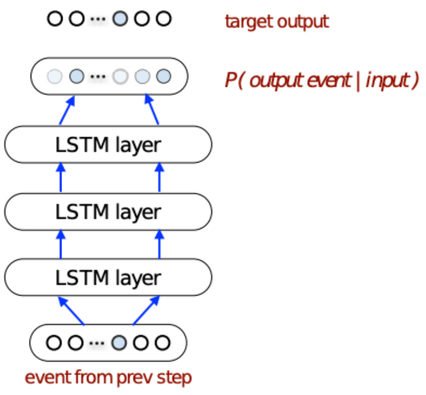


Figure 2. network architecture. Input is a list of list of embedded pitches, output is a categorical
[5]

Finally, we chose the Categorical Cross Entropy Loss a loss function. As a first shot, we thought that the mean squared erros loss could have been a good choice since the embedding is done with float integers. The issue with is that the vocabulary (that is to say the number of different pitches) is very large (about 10 thousand combinations in the dataset, which is more than the number of french words currently used by individuals). Hence, the predicted value is almost all the time not correct, and using this loss would only underfit what is to learn. Using the Categorical Cross Entropy Loss is more adapted with our problem of classification with more than 2 classes. And to optimise our network we will use a RMSprop optimizer as it is usually a very good choice for recurrent neural networks.

The formula of the used loss function is the following:

$$L_{y'}(y) := -\sum_i y'_i \log(y_i)$$

with y'_i the correct output and y_i the prediction for sequence i out of n sequences.

3.4. Hyperparameter tuning

Hyperparameter selection is a challenge in this project. To get the best combination for our specific model, we created various ranges for each hyperparameter and observed the loss function and the training for each one. The hyperparameters we identified are the following:

- sequence length
- amount of hidden layers
- amount of dropout
- batch size
- number of epochs

3.5. Evaluation

An objective evaluation of the model is quite challenging for what concerns music generation problem. The loss function can be used to measure the difference with the original song. Validation set is built using the predefined Keras function and set at 20% of the overall dataset.

Excepted the case when a small pattern is fed recursively in the neural network as we discuss it in next session, we cannot expect the loss function to reach a null value. Indeed, that would mean that we are able to predict exactly what the artist will play given the beginning of his song. Since music is not based on strict rules and because the embedding was done with an arbitrary rule, predicting closely what is going to be played is impossible.

Another way to evaluate the quality of the model is by listening the generated sample on the test set. To do that, the embedded outputs have to be converted into a MIDI file. The time between each pitch was defined arbitrary (we chose 120 pitches per minutes). What was proposed by [5] is to get ”feedback from professional composers and musicians”.

4. Experiments

In this section, results of experiments are provided. The model training and testing was done on Google Colab with GPU as a backend.

4.1. First shot: learn recurrent patterns

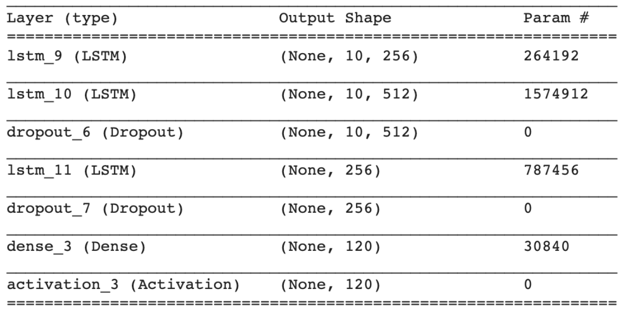
4.1.1 Dataset

As a starting point, we decided to train the neural network with a recurrent pattern. The input of the network consists of a series of 240 pitches. 240 pitches corresponds to 2 minutes of sound. We feed 200 times this series into the network.

The reason why we decided to train the network and generate music with this data is to

- test the consistency of the network
- verify that in a perfect situation, the structure we provided is able to learn the rules

4.1.2 Results of training phase, fine tuning



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 10, 256)           264192
_________________________________________________________________
lstm_10 (LSTM)               (None, 10, 512)           1574912
_________________________________________________________________
dropout_6 (Dropout)          (None, 10, 512)           0
_________________________________________________________________
lstm_11 (LSTM)               (None, 256)               787456
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               30840
_________________________________________________________________
activation_3 (Activation)    (None, 120)               0
=================================================================
```

Figure 3. Network architecture used

The fist parameter we wanted to calibrate is the length of each sequence provided to the network. Below are the results of the experiment. We measured the number of epochs necessary to reach a validation loss of 0.01.

What can be observed here is that the longer the sequence, the easier it is for the model to converge. In the other experiments, we will keep the sequence length at 50.

Additionally, we have tested the depth of the architecture and the amount of dropout.

hidden layers	epochs needed for validation loss=0.01
1	12
2	5
3	6

Table 1. Influence of hidden layers amount

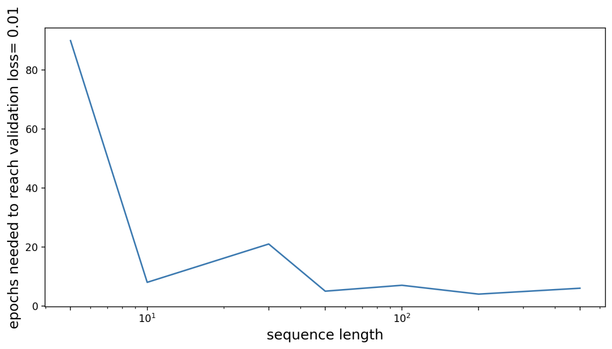


Figure 4. Sequence Length

dropout	epochs needed for validationn loss=0.01
0	5
0.1	5
0.2	5
0.3	5
0.5	5
0.8	12

Table 2. Influence of dropout

With our dataset, having 2 or 3 hidden layers is better than only one. The depth can be set at 3.

Regarding the dropout, a too high value unsurprisingly not a good option. Here, we cannot really see the advantage of this regularization method because of our data structure. We will keep it at 0.3.

Batch size is also something we could play with. A smaller batch size takes more time to be computed but it has also a regularizing effect. Thus, we observe a lower generalization loss.

batch size	epochs needed for validation loss=0.01	training time
32	3	8.6 min
128	8	6 min
252	10s	4.2 min
512	16	3.8 min

Table 3. Influence of batch size

Last parameter we might want to test is the impact of number of epochs on loss. Results are on figure 5.

One interesting fact that can be seen here is a spike on the validation set loss. That is why we used the Checkpoint feature of Keras to only save best model at each epoch.

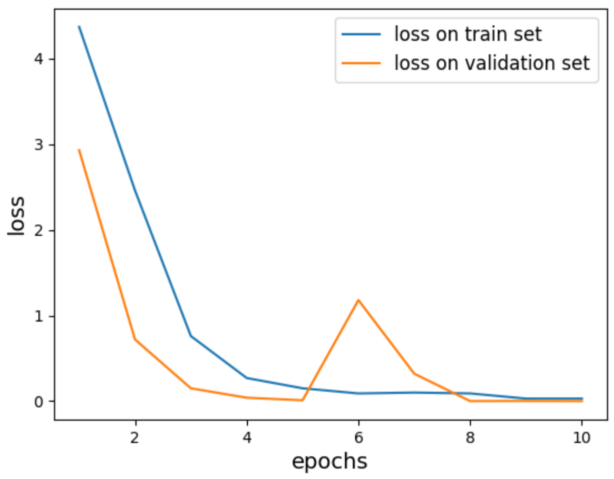


Figure 5. Loss function on the training phase

4.1.3 Music generation

As a test set, we randomly took 50 notes from the vocabulary used in the training phase. This is the starting point for our model to generate notes.
Input can be listened here: `https://soundcloud.com/gambino_gregoire/generated-song-1st-generation`. Since it was randomly generated, you might not be surprised that it does not sound perfectly good
Here is the generated output: `https://soundcloud.com/gambino_gregoire/generated-song-1st-generation`. This one is better than the random input we provided to the network. We might think that the neural network took some benefits from the training phase in the dataset extracted from professionals.

4.2. Second shot: feed the network with more data

4.2.1 Dataset

This time we trained the neural network with more data, coming from several different songs. We artificially augmented the data by duplicating sequences of the clips. The hyperpaparemeters used were the same as the one defined in previous subsections

4.2.2 Results on training, fine tuning

Below (6) is the loss function of the model. What can be added to that curve is the training time, significantly higher than the one of previous phase since the amount of inputs is higher and much more complex.

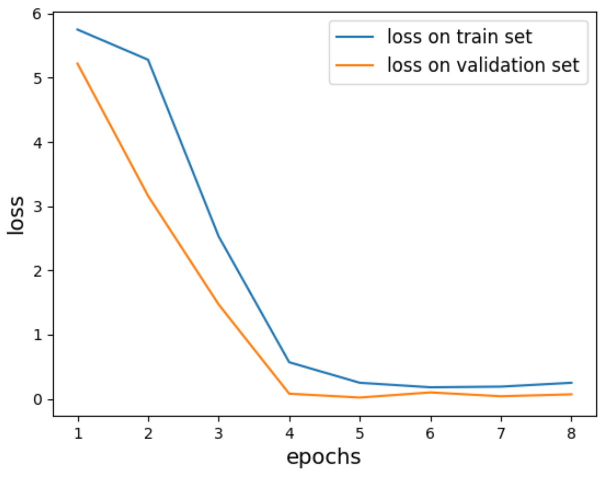


Figure 6. Loss function on the training phase

4.2.3 Music generation

The same methodology as before was taken: 50 notes were randomly taken from the vocabulary used in the training phase
Input can be listened here: `https://soundcloud.com/gambino_gregoire/input-2nd-generation-process`.
Here is the generated output: `https://soundcloud.com/gambino_gregoire/generated-song-2nd-generation`. This one sounds pretty good and less artificial than the one generated before. Once again, the more data we feed into the network, the better the sound quality.

5. Conclusion

5.1. Critics of our work

To conclude, we have seen that our music representation and our LSTM based network gave some promising results when we trained our model with a small number of notes and simple patterns. Indeed, the resulting melody that seems musically consistent.

However, our model faces some challenges when we tried to use more songs of our data set. We believe that it is mainly due to the embedding of the notes we have chosen. Indeed, we end up having a giant vocabulary (around XX notes), which we can compare to the French language that has XX words. In that effect, since our calculation capacity is very limited, our network had not enough inputs regarding to the number of outputs possible. A representation of the notes in a vectorized way might have been way more relevant.

Additionally, the dataset we have used is also very wide and diverse. The songs are very complex, and some are very different. Given that our calculation capacity, we only have been able to use a few of them. Thus, it would be more consistent to pre-select some training songs of the same style so that the network is able to identify common patterns.

5.2. Future work

As explained above, it would a vectorized embedding of the pitches can be more consistent as it would result with a hardly smaller vocabulary. It would also be interesting to add to the model the duration of the notes as well as their loudness to obtain a more human-like melody.

Moreover, even though LSTMs show good results, State of the art techniques now focus on Transformers structure [6] [3]. Unlike RNNs which computes sequentially the data, transformers can parallelize calculations in a much more optimized way and we could exploit more inputs with the same machines. Additionnaly, transformers tends to perform better than LSTM networks because of theur ability to have ac-cess to any earlier event. In our music generation problem,where rules and causality is not time fixed, the flexibility ofthis computing costly structure is powerful.

References

[1] School of Music, University of Minnesota. `https://www.piano-e-competition.com/`.

[2] B. Bozhanov. Computoser - rule-based, probability-driven algorithmic music composition, 2014.

[3] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck. Music transformer. `https://arxiv.org/pdf/1809.04281.pdf`, 2018.

[4] J. Maurer. A Brief History of Algorithmic Composition. `https://ccrma.stanford.edu/~blackrse/algorithm.html`, 1999.

[5] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan. This time with feeling: Learning expressive musical performance. `https://arxiv.org/pdf/1808.03715.pdf`, 2020.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. `https://arxiv.org/pdf/1706.03762.pdf`, 2017.

[7] Wikipedia contributors. Musikalisches würfelspiel — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Musikalisches_Wurfelspiel`, 2020. [Online; accessed 20-February-2021].