23.06.2019, Gliwice

Subject:

## Biologically Inspired Artificial Intelligence

Project:

# Generating music using LSTM neural networks

| | |
|---|---|
| Autor: | Grzegorz Kazana |
| Semester: | VI |
| Group: | GKiO3 |
| Section: | 7 |
| Tutor: | dr inż. Grzegorz Baron |
| Link to repository: | https://github.com/GrzegorzKazana/LSTM-music-generator |

## 1. Introduction

The core idea behind this project sprouts from one question: “Can computers be creative?”, or in other words, are they able to not only process data, but also create new meaningful information? Standard applications of neural networks consist of classification, clustering and regression, which provide great value and solve problems on unimaginable scale, but do not manifest creativity. An example of a task, that requires talent and inventiveness is composition of music.

## 2. Methods and implementation

Music, according to Wikipedia, is an art form which medium is sound organized in time. It means, that in order to reason about music, we have to keep in mind two things: what sound is being played, an what is the succession of sounds. It implies, that music is sequential data.

In order to employ machine learning in context of music, we will assume that it is possible to infer the next musical sound knowing the preceding sounds.

### 2.1 Neural network structure

Since we are dealing with sequences, recurrent neural networks are the most obvious choice. From the broad selection of RNNs, I decided to use LSTMNN (Long Short-Term Memory Neural Network), which solves issues concerning classical RNNs, like vanishing gradients.

### 2.2 Data

Usage of real audio files (like .wav), given their size attributed to sampling rate in the order of tens of thousands and non-obviousness caused by presenting only sounds amplitude, was not a viable option. With that in mind, I decided to use MIDI files, which encode music in simpler (but also more limiting) way. MIDI files consist of tracks and channels, which contain information of specific musical notes played over time. Each event in midi file (like note on, note off or tempo change) is encoded as a message emitted in specific time.
For simplicity, I picked single instrument pieces (piano) composed by Beethoven.

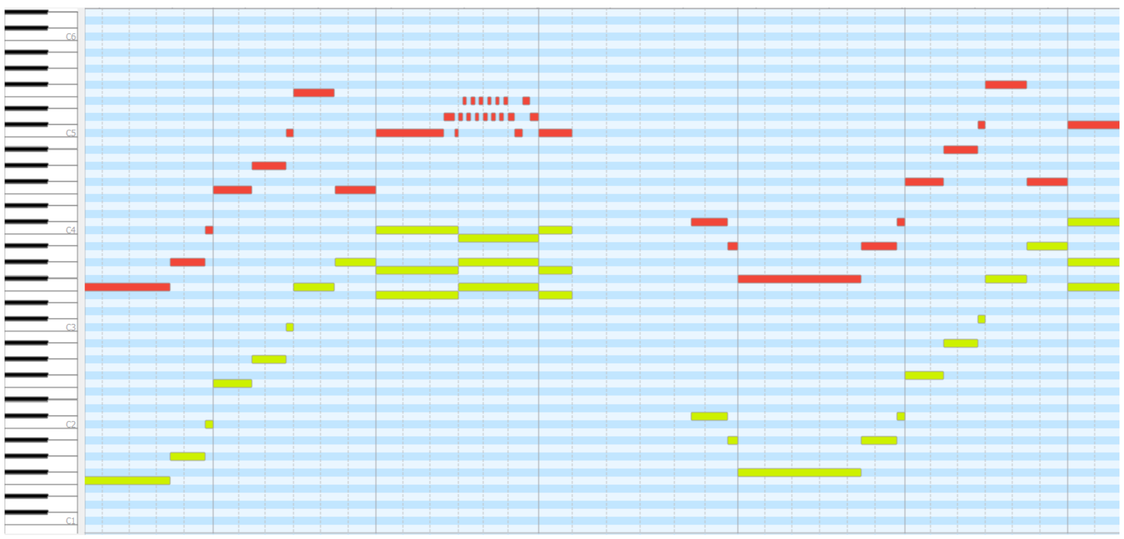


Fig. 1. Exemplary midi file – Sonata No. 23 F minor

## 2.3 Data processing

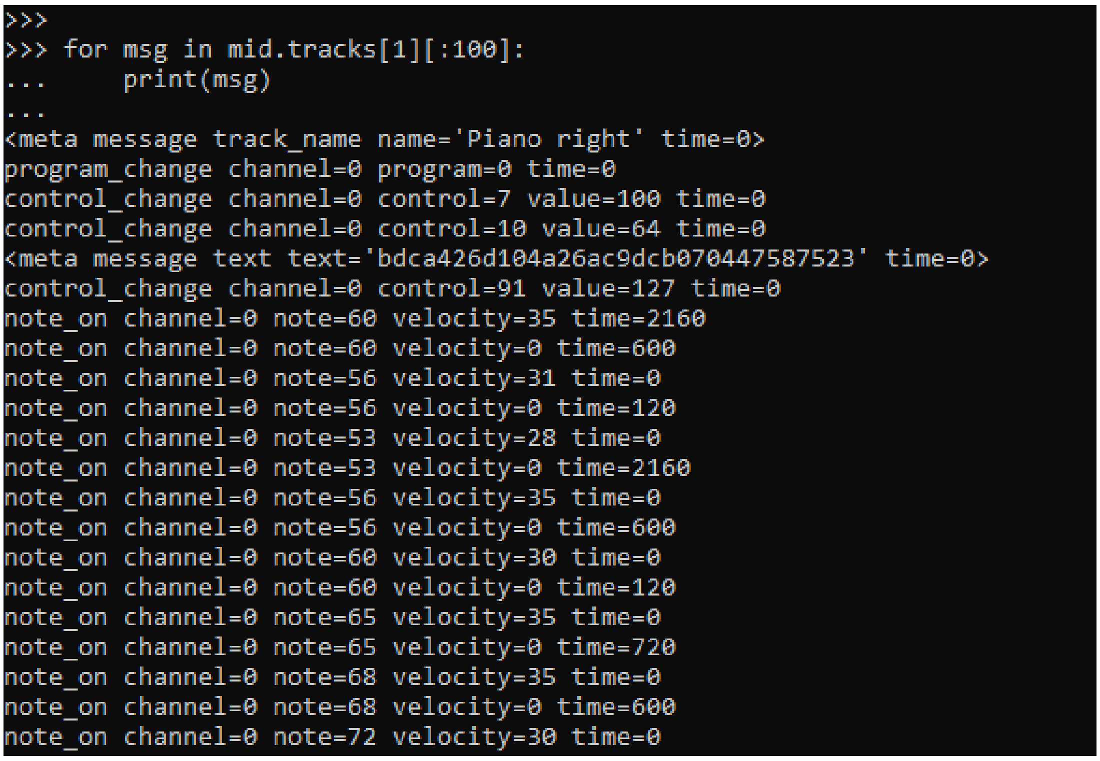
In order to read midi files, to turned to established and tested solution – Mido python library. It allowed me to access midi note_on and note_off messages and their times in a simple manner. The next task was transforming that data into matrices – the only format understood by neural networks.

First, I decided to quantize time to 100ms chunks, which means that a single vector of notes will represent a smallest instance of time. The choice of time chunk length was a compromise between lowering dataset sequences length, making the leaning easier, and keeping the music accurate. The caveat of such choice is that, the representation is not ideal – notes shorter in duration than 1/16s (assuming standard tempo of 120bpm) are not faithfully represented. Chunk length that seemed to conserve almost all noticeable resolution was 25ms, but with longer sequences, learning suffered - relations between notes were "stretched" in time, making them much harder to be learned.

The second question arises, when it comes to encoding notes. Midi format uses 7 bits to encode pitch, giving 128 possible notes. We could encode notes as vectors of 7 binary values, however it is not the best solution. First of all, we wouldn't be able to represent instances, when two notes are pressed in the same time. Another issue is, such format implies that notes 75 ($1001011_2$) and 11 ($0001011_2$) are really similar, more similar in fact, than 75 and 87 ($1010111_2$) which is the same note just octave away. As a solution to that, I decided to encode notes as 1-out-of-128 vector (note 75 would be represented as a vector of 74 zeros, one, and 53 zeros). Such solution allows for encoding chords (multiple notes sounding in the same time) at the cost of increased input dimensionality and storing lots of zeros.

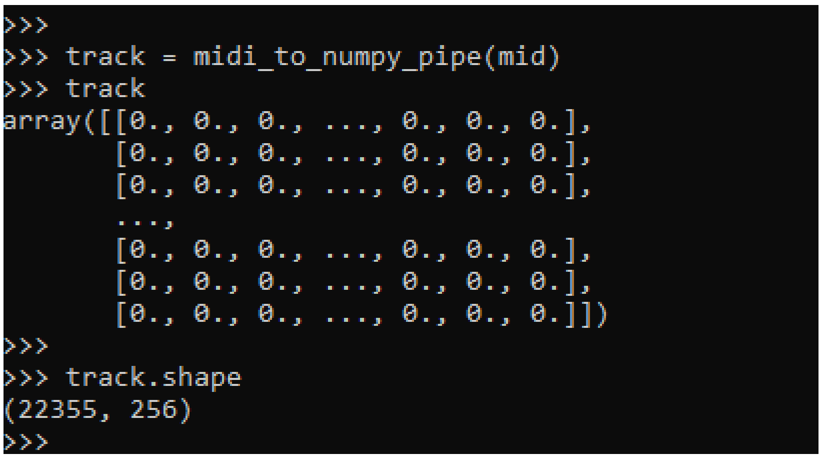
Above choices imply, that 30 second musical piece would be represented as a 300x128 matrix. Processed midi files were saved as scipy.sparse matrices for maximum storage efficiency.

Another dataset augmentation I employed was musical transposition across 2 nearest octaves, which means that pieces were copied to another keys. Logic behind that is, relations between notes are much more important than sequences of notes in original key. It also gives us bigger dataset.



```
>>>
>>> for msg in mid.tracks[1][:100]:
...     print(msg)
...
<meta message track_name name='Piano right' time=0>
program_change channel=0 program=0 time=0
control_change channel=0 control=7 value=100 time=0
control_change channel=0 control=10 value=64 time=0
<meta message text text='bdca426d104a26ac9dcb070447587523' time=0>
control_change channel=0 control=91 value=127 time=0
note_on channel=0 note=60 velocity=35 time=2160
note_on channel=0 note=60 velocity=0 time=600
note_on channel=0 note=56 velocity=31 time=0
note_on channel=0 note=56 velocity=0 time=120
note_on channel=0 note=53 velocity=28 time=0
note_on channel=0 note=53 velocity=0 time=2160
note_on channel=0 note=56 velocity=35 time=0
note_on channel=0 note=56 velocity=0 time=600
note_on channel=0 note=60 velocity=30 time=0
note_on channel=0 note=60 velocity=0 time=120
note_on channel=0 note=65 velocity=35 time=0
note_on channel=0 note=65 velocity=0 time=720
note_on channel=0 note=68 velocity=35 time=0
note_on channel=0 note=68 velocity=0 time=600
note_on channel=0 note=72 velocity=30 time=0
```

Fig. 2. Midi file messages accessed using Mido library



```
>>>
>>> track = midi_to_numpy_pipe(mid)
>>> track
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
>>>
>>> track.shape
(22355, 256)
>>>
```

Fig. 3. Midi file transformed to numpy matrix

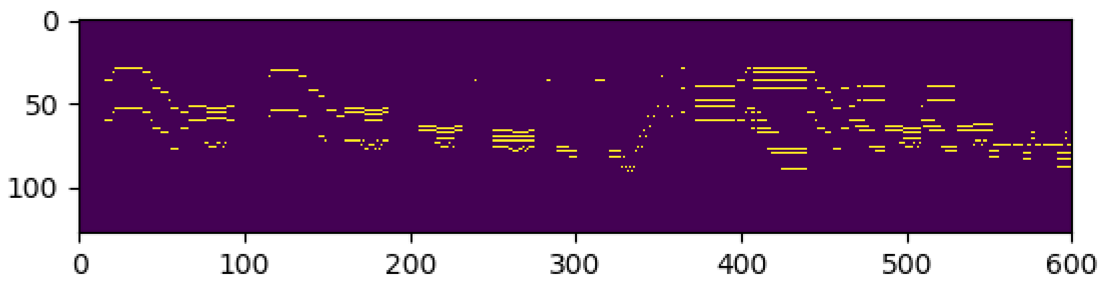


Fig. 4. Matrix plotted

## 2.4 Training method

We are going to train neural network to learn relations between notes across time. This could be treated as a classification task. Given n previous time steps, we will be trying to teach then network to predict n+1 time step.

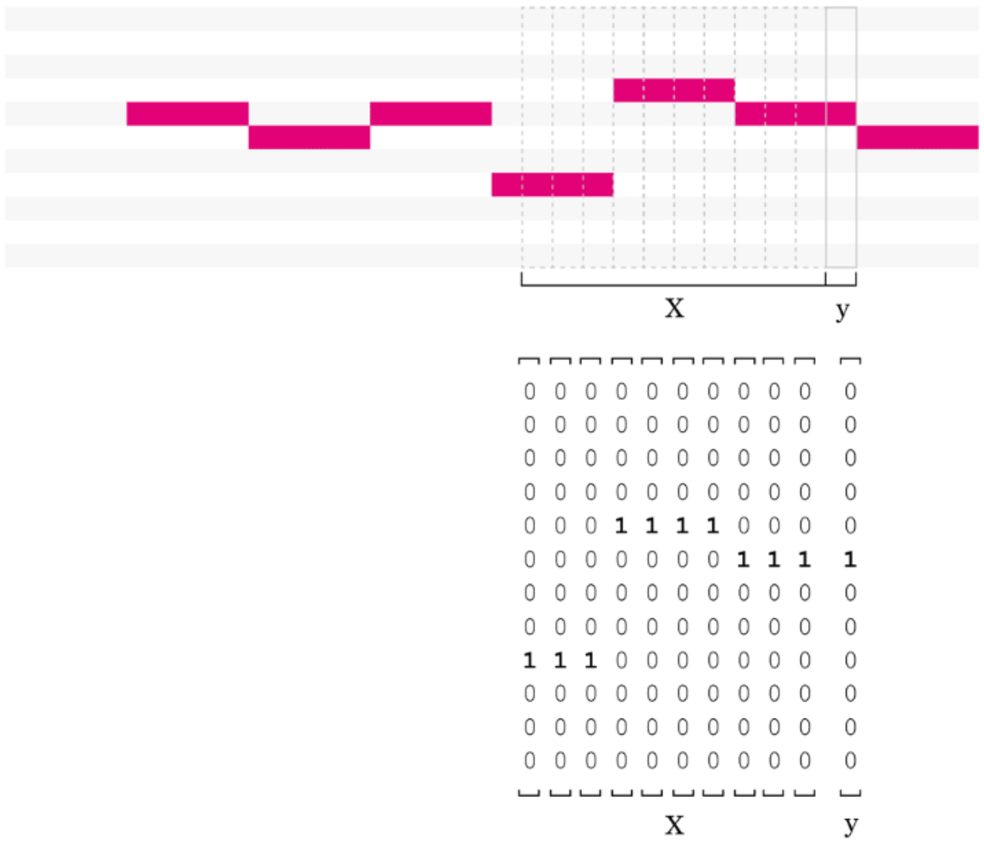


Fig. 5. Illustration of input sequence X and target y

As for selecting data samples, using whole tracks as sequences would not be effective – even LSTMs have their limits, and processing such long data streams would not be optimal. Instead, I decided to pull sub sequences of tracks, in a windowing fashion. The window size I choose was random, and varying between 20 and 100 timesteps – corresponding to 2-10s time slices. Values were selected on presumption, that musical context – track tempo, key and current tension could be inferred from samples of that length. In retrospect, I could have been more thoughtful about this parameter, and should have tested the performance when using longer sequences, possibly giving the neural network more insight into the past.

## 2.5 Model

I choose to use LSTM implementation found in tersorflow high level API – keras. Model's input and output size obviously matches the selected dataset dimensionality – 128. Number and size of recurrent layers was selected by trial and error. Experiments with smaller dataset and 256 perceptron single LSTM layer gave good results. For whole dataset I stuck with 2 recurrent layers of sizes 512 and 256 respectively. As our labels (predicted notes) are multiple binary features, I used Binary Cross Entropy loss function. For optimizer I choose Adam, as it solves most issues of adaptive optimizers like Adagrad.

## 2.6 Training

Model was trained on Google Colab platform, utilizing GPU acceleration, resulting in x30 speed increase in comparison to my laptop CPU. Model was trained for around 12 hours.

## 2.7 Generation

Now, for the major topic of this project – music generation – the reasoning was: if the neural network is trained to output next musical note based on previous ones, then by appending the predicted note to the input, we could potentially repeat this process ad infinitum and therefore obtain unlimited, hopefully original and decently sounding samples.

The subject of great importance, was choosing the way to start the generation. I tested various methods, including random noises, silence and random melodies in and off key.

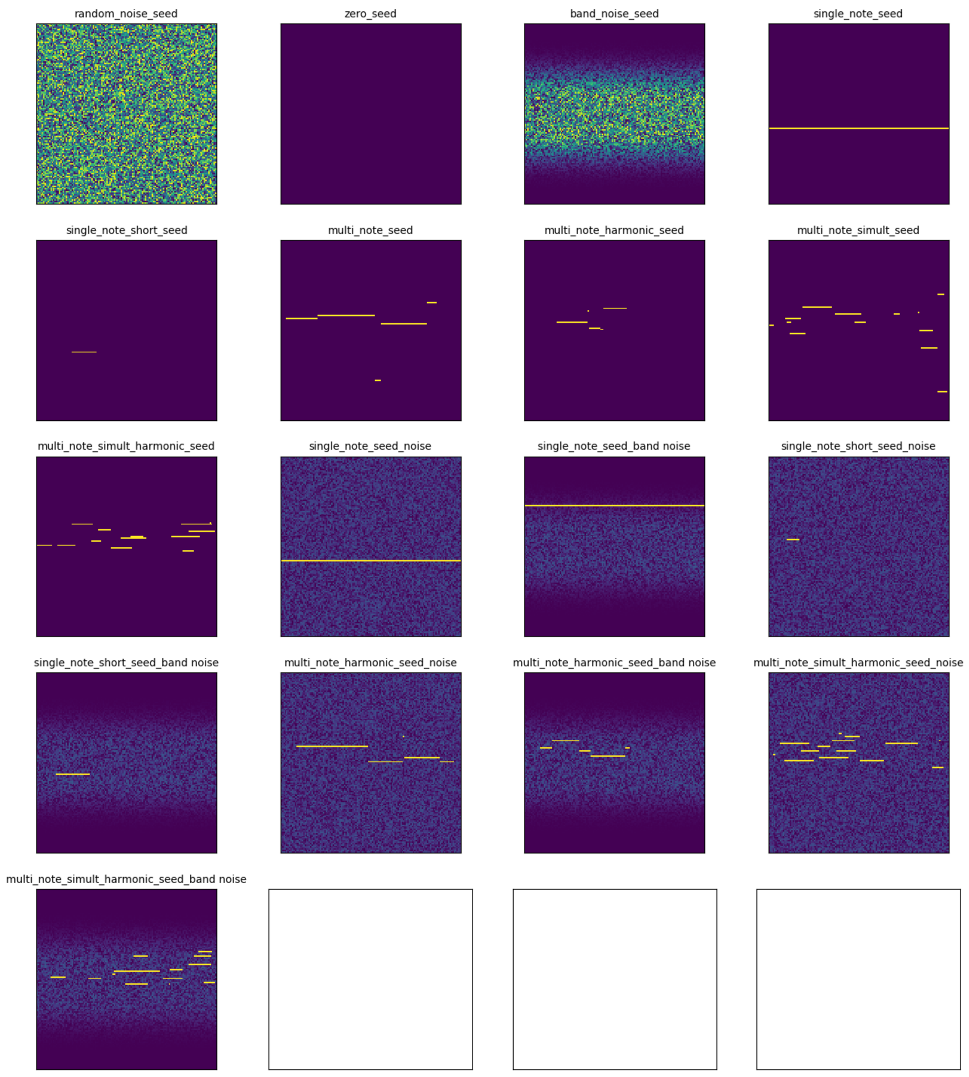


Fig. 6. Visualisation of 17 seeding methods

## 2.8 Results

With each generating method, I generated a batch of samples. Each seed was 5s or 10s in length, meaning the first 50 or 100 timesteps were not created by neural network. Generated samples were 30s long.

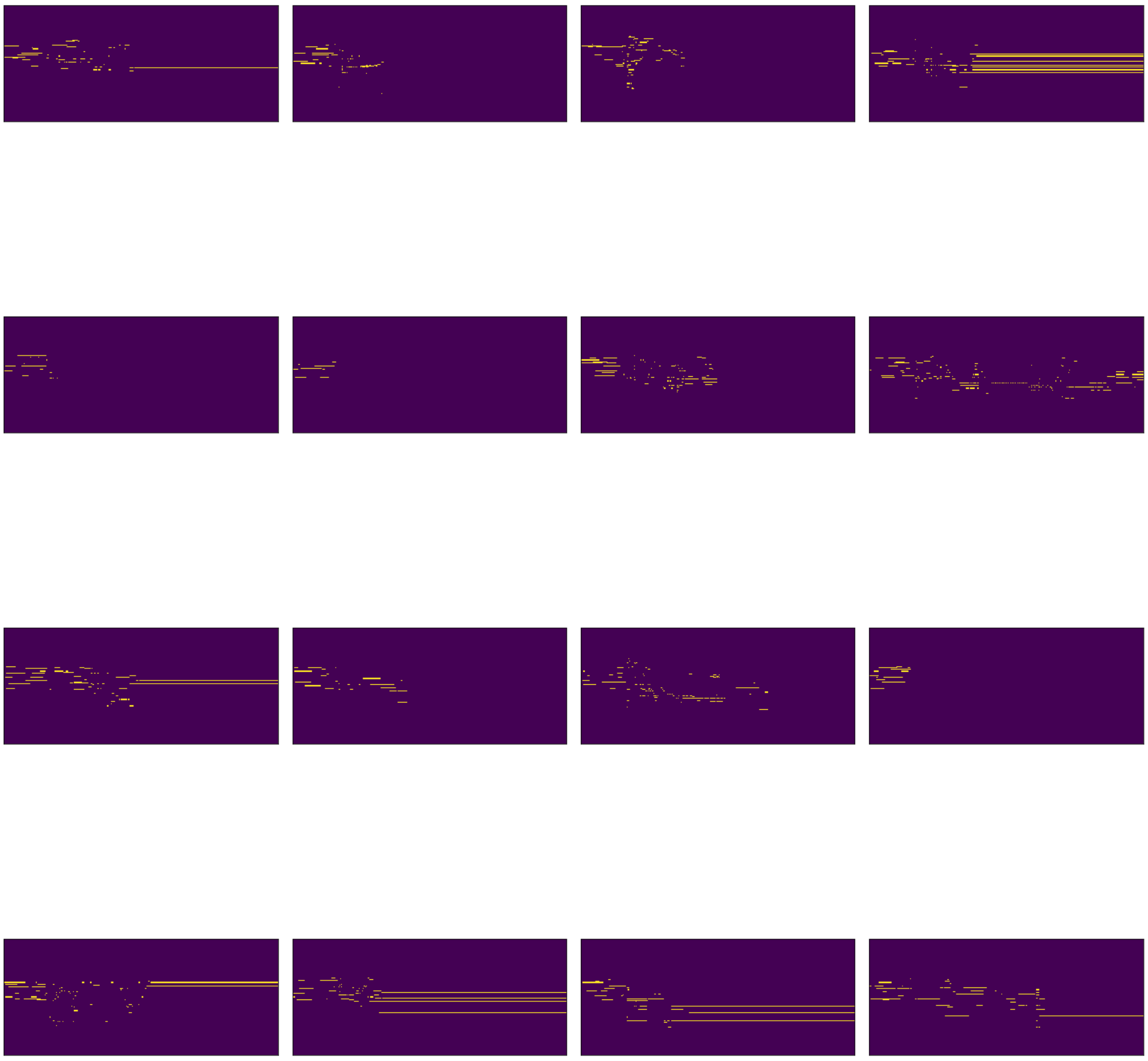


Fig. 7. 16 samples generated using simultaneous multi note harmonic seed with band noise

Samples were then converted back to midi format, and manually listened through. The most interesting samples were:

- *m1560173495_multi_note_harmonic_seed_noise_2019-06-11_19_05_39.460937_14.mid* – In this sample the network seemed to utilize its knowledge of rhythm – as the tempo of the track to be cohesive in its whole length (apart from first 5s - the seed), and also the knowledge of harmony – notes played were also coherent, and only in the end started to feel a little off.
- *m1560173495_random_noise_seed_2019-06-11_19_45_55.587066_8.mid* – Generated using random noise. Tracks generated using this method seem to have one thing in common – a pompous beginning and hectic style.
- *m1560173495_multi_note_simult_seed_2019-06-11_18_55_34.627148_14.mid* – Generated using simultaneous multi-note seed. One of the most interesting results, after being seeded, the network continued the sample with more or less random keystrokes, but eventually started playing the intro to "For Elise", which was obviously in the dataset. This suggests that the network could have overfitted the dataset, or that the melody found in "For Elise" was either recently used in training, or was occurring in the training dataset more frequently. I lean toward the latter explanation, on grounds that network was not really eager to continue the melody when presented to the original (samples *m1560173495_elise_0_10sec_seed_2019-06-20_15_30_35.200686_0.mid* and *m1560173495_elise_60_70sec_seed_2019-06-20_15_39_49.918648_0.mid*), and the main motif in that piece is repeated multiple times, making the model exposed to it more often.
- *bach_846_0_10sec.mid* – Generated using the beginning of Bach's Prelude and Fugue in C major. The network failed to continue the seed, which could either mean the model decided the sample was already complete, or that it was too unfamiliar with the input and applied the safe approach – remaining silent.

## 3. Conclusions

Although, the results are acceptable, they are far from being indistinguishable from human creations. The reasons may be:

- Selected data format was not optimal for training –
    - Representing time as a quantized series of frames, leads to inefficient representation of notes sustained for longer periods of time (e.g. single note held for 2s is represented as 20 identical vectors). An alternative could be using musical notation – in which the note would be represented as a vector containing additional values representing the length (e.g. quarter-note, 1/16th…). Unfortunately, such approach has its caveat, it would be necessary to define a set of available note lengths, what in more complex pieces could be quite large.
    - Also, the encoding of note pitch as 1-of-128 could be argued. Such format eliminates false relations between notes explained in 2.3, but it also is not exposing true relations, like octaves or intervals between notes of the same scale, which could have positive effect on learning.
    - Furthermore, if we were able to move another piece of work to the pre-processing stage, and feed the network information about degrees of scale rather than absolute notes, it could also greatly improve the performance.
- Hyperparameters' values were not properly tuned. The parameter which could greatly affect the learning process is the window length. Setting it to a higher value could make the learning harder, but would provide larger context for note prediction.
- The model choice and structure could be revised. While the size of the network seemed sound, other models, like Generative Adversarial Networks, could turn out to be better choice.
- The dataset itself was hard, after all we were training to train a neural network using Beethoven's art pieces.

## 4. Sources

- http://www.piano-midi.de/
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://arxiv.org/ftp/arxiv/papers/1804/1804.07300.pdf
- https://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neural-networks/