Using LSTM recurrent neural networks and audio files for automatic music generation

(MP 093 GroovyBee)

Mohammad Omar Nasir (663890) Laura Deleuze (604642)

January 31, 2018

Abstract

This report explores the possibility of using raw audio samples, digitized as Midi files, to train a Long Short Term Memory recurrent neural network (LSTM). LSTMs are the popular choice for handling time-series datasets, and are therefore well suited for audio files. Previous work in the field includes variety of music data, including musical notes represented as both sequence of characters and numerical vectors. Although, they seldom address the polymorphic aspect of classical music. Therefore, the content of the report outlines various methodologies adopted to process the raw audio dataset into suitable format for the LSTM network training, and implementation of a custom architecture for the end-to-end LSTM network. Dataset used in this project involves extraction of instrument level data from a MIDI track, in order to ultimately discover the effectiveness of learning an univariate distribution. Eventually, the generated succession of notes are compared with the pattern of musical notes in existing Midi files, to understand the effectiveness of the output of the network. However, using digitized audio samples does not seem to offer the expected improvements over transcribed audio data containing musical characters. Though, cascading of neural networks that individually learn and track changes in tempo, notes from various instruments, and silence periods may lead to better results in terms of musical integrity and composition.

1 Introduction

There is no denying that music is an intellectual activity as stated by the musicological research community. Music, in the sense of the ability of recognizing patterns, imagining them and actively modifying them, is even considered as the essence of the human mind [1]. Indeed, E. R. Miranda explains in his book that musical activities, including music generation, involve complex memory mechanisms, both conscious through the concept manipulation and unconscious through the access to the dense neural network that constitutes not only the brain but also the whole human neural system [1]. As a result, studying music activities, in particular how humans create music, would significantly enhance our current understanding of the human brain behaviours. In other word, such research would significantly contribute to the development of a real Artificial Intelligence (AI). In addition, music activities heavily rely on human memory mechanisms that Artificial Neural Networks were conceived to and are now relatively able to mimic. Therefore, this research study addresses the issue of how to automatically generate music. We chose to exploit a Long Short Term Memory recurrent neural network (LSTM RNN) for its memory-like features. We trained it

on audio files (MIDI formats) of classical music songs as music is not only about a succession of notes but also about how the musicians plays it in term of pitch and rhythm. Besides, classical music is often governed by strict rules defining patterns that we assumed neural networks may easily recognize and learn from. The ultimate aim of this study is to train a LSTM RNN how to generate new coherent music MIDI files luring people to think it was composed by a human.

2 Related work

Automatic music generation is not a new research field. Indeed, Research publication about music composition by computers can be traced back in 1957 when Hiller and Isaacson developed the first Illinois automatic computer (ILLIAC), able to generate music owing to mathematical algorithms [2]. Besides, C. H. Liu et al. listed several attempts and studies to compose harmonious melodies in the past decades [3]. Since Howe attempt in 1975 to codify musical elements into multidimensional arrays in order to determine pitches and rhythms locations through distinct operators to eventually generate music [4], C. H. Liu et al. identified a significant trend among researchers to use Deep Learning, in particular recurrent neural networks, to create new music [3]. Indeed, analysing music data, modeling sounds and generate new songs becomes theoretically easier owing to the increasingly easy access to resources such as computation power, data and advanced training methods like artificial neural networks. For instance, Strum et al. used a LSTM RNN to compose folk music transcriptions. Their study showed how efficient LSTMs are to learn and create folk music as this genre is heavily based on the recombination of familiar patterns [5]. Although such characteristic reveals LSTMs to be relevant in the study of Classical music, Strum et al. exploited note transcriptions of music only, in the ABC format [5]. Therefore, they are missing the whole audio performance around one music track, outside the note transcription itself.

As Shuvaev et al. highlighted in their study, expert listeners are able to recognize musicians and composers while listening to few seconds of a track [6]. Based on such observations, they built a Convolutional Neural Network classifier training on audio recordings of classical music from different composers instead of their sole transcriptions. Their music representation via Random Matrix Transform (RMT) yields promising results in classifying as well as generating new music [6]. Likewise, Wavenet, a deep learning based generative model, showed strong abilities to generate realistic musical fragments based on raw audio waveforms [7]. Thus, neural networks appear to be perfectly able to capture relevant musical characteristics out of audio files. As a result, researchers turned to combine LSTM and audio files in MIDI format: I. Liu et al. took a time series approach to relatively successfully reconstruct Bach compositions with LSTM combined with Resilient Propagation (RProp) [8].

However, music is more than just a time series problem: the polymorphic dimension is non-negligible, even in orchestra cases. Huang et al. proposed to model this polymorphic aspect with RNN combined with Restricted Boltzmann Machines. They translated the MIDI files in character level concepts in the aim to quite efficiently capture the music melody and harmony and generate human-like composition. However, they also observed how learning long-term music structures was lacking in their experiment, constraining their final performance [9].

3 Method

Recurrent Neural Networks (RNN) have been popular for handling time dependent data. RNNs produce an output at timestep t, which is combined with the input at the next timestep $t+1$ to generate the next output. In practice, however, there is a drawback to using RNNs for large time-series. Theoretically, the network exists in a loop; each state of the network only receives information from the previous state. Hence, an RNN would not be able to retain meaningful patterns beyond 2 time-steps. Music is composed of inherently complex structures that exist as a function of time, therefore, a variant of RNN known as LSTM is more suitable.

Fundamentally, LSTMs possess the capability to retain longer periods of input in their memory. LSTMs achieve this retention by introducing the idea of cell states. Cell state is a transformation from input to output. This can take place linearly, or the network can manipulate the input data inside the cell. The manipulation is achieved via the inclusion of LSTM Gates, which are briefly explained below (Figure 1):

1. **Forget Gate:** The function of Forget gate is to exclude certain parts of data at timestep t. This is achieved by using a *sigmoid* function, that produces a range of outputs from 0 (implying discarded data), to 1 (data to include).

2. **Input Gate:** This gate functions to include input data at t selectively. This is achieved by applying *sigmoid* and *tanh* functions to it, separately. The *sigmoid* function determines the data point to be retained, while *tanh* determines how it is to be included.

At this stage, the new data at timestep t is passed through both Forget Gate and Input Gate respectively. The output of the Forget gate is **multiplied** with the Cell State from timestep $t-1$, and later on **added** to the output of the Input gate. Essentially, this removes certain parts of the data, while adding new information selectively.

3. **Output Gate:** The Output gate applies a *sigmoid* function to the input data, while the current Cell state is passed through a *tanh* function. The output of both operations is multiplied to selectively finalize the portions of output needed to be produced by the network.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1+b_f} \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1+b_i} \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1+b_o} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Where f, i and o represent the forget, input and output gate respectively. c and h represent cell and hidden states.

Holistically, the neurons of the hidden state associated with each Gate are trained via back-propagation, while the current cell state is updated in the forward pass. Theoretically, given new data at timestep t, both the hidden and cell state would essentially retain memory of data that was fed to the network at

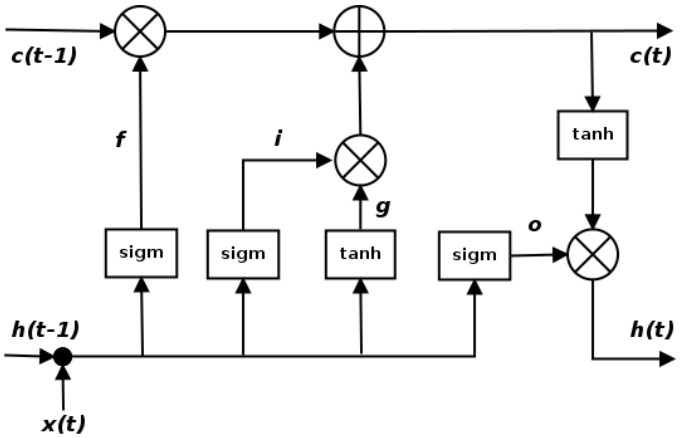

Figure 1: Detailed LSTM cell visualization.

$t-1$, $t-2$, $t-3$, and so on. Hence, it is evident that LSTM offer significant gains over the quality of the output, as compared to traditional RNNs.

In this project, a 2-Layer LSTM network was implemented, including dropout layer for regularization, and a fully connected final layer to transform the output to the required vector size. The feature vector of MIDI files includes a total of 128 pitches, hence the input to the network was a time series of 128 x Sequence Size. The dimensions of the first LSTM layer were 128 and 512, respectively. This output was fed into Dropout Layer with probability p, which was then fed into the second LSTM layer with dimensions 512 for both input and output. Finally, this output was fed into the fully connected layer with dimensions 512 and 128, to transform the predicted vector into 128 pitches. The loss function used was Mean Squared Error loss.

Various hyperparameters include the dimensions of the hidden layers (512), dropout regularization, Adam gradient descent learning rate, Batch size, Epoch size, Sequence size, and step size. The sequence size is the length of the subset of a MIDI track, in units of time, while the step size determines the starting point of the subsequent sequence. For example, a sequence of 10 would take samples from $t = 0$ to $t = 9$, while a step size of 5 would ensure the second subset of the data to be fed to the network would begin at $t = 5$ up to $t = 14$. For each sequence, the corresponding test vector was taken to be the 11th sample. The loss function compared it with the prediction based on the 10 samples in the sequence. The dimensions of the transformed input data were calculated as:

$$input\ range = \frac{track\ length - sequence\ size}{step\ size}$$

To generate music, a random subset of the input data was selected and fed to the neural network. The prediction was fed cyclically into the neural network to continue generation of musical notes, for a certain fixed number of timesteps.

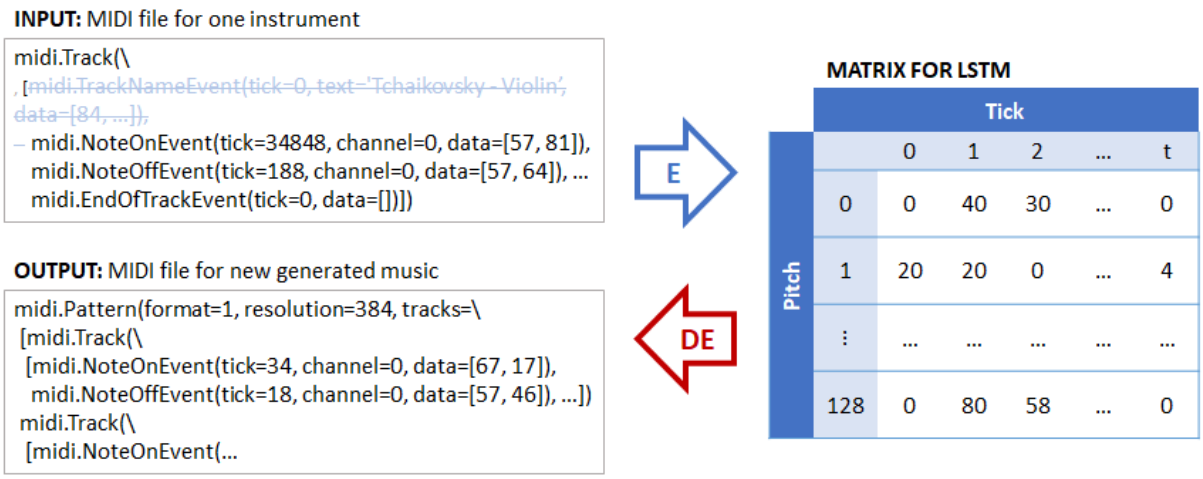

Figure 2: Simple model illustrating the encoding (E) and the decoding (DE) of MIDI files into the 2D matrix used by our LSTM.

4 Data

For this paper experiment, 132 audio files of three famous classical composers - Beethoven, Mozart and Tchaikovsky - were downloaded from the website **midiworld**, in the format of MIDI files.

MIDI is an industry standard music technology protocol that transmits real-time information to reproduce a piece of music on different synthesizers and computers [10]. MIDI structured language encodes songs into a succession of events defining what kind of sounds should be played and when. *NoteOnEvent*, *NoteOffEvent* and *EndOfTrackEvent* are the note events we considered for this experiment. The time is measured in *ticks*, on tick representing a fraction of a beat or a quarter note.

However, LSTMs cannot efficiently learn from full audio, thus whole MIDI files. Therefore, we preprocessed the MIDI files as inspired by Weel relevant encoding and decoding methods [11]. Figure 2 shows how we preprocessed the MIDI files to first encode our data: we (1) extracted from one audio each instruments track separately, (2) removed all metadata not related to MIDI notes events as well as the silence breaks to ensure a time continuity and (3) we transformed each output into a 2D matrix with the 128 possible notes as one axis and the tick numbers as the other, attributing their corresponding pitch intensity (velocity value). Afterwards, Figure 2 illustrates how we decoded the generated new individual instruments tracks, combining them back into one single unique audio: (1) we translated the instrument matrices vector by vectors into the corresponding MIDI notes events, appending them to their corresponding MIDI track; (2) we merged the tracks into one MIDI file. Both encoding and decoding algorithms were developed using the Python-MIDI toolkit [12] to access the MIDI files content with ease.

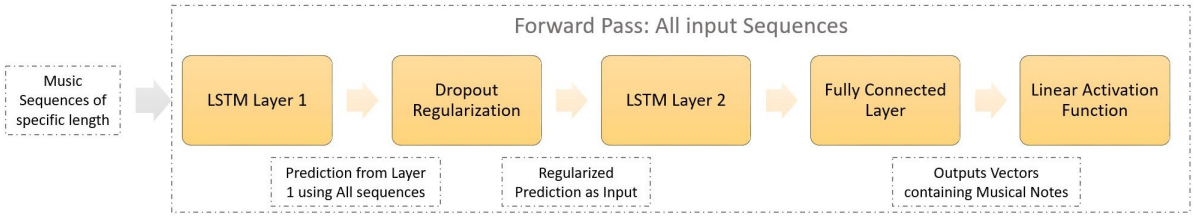

Figure 3: Neural Network architecture using built-in LSTM PyTorch Modules

40	39	41	34	42	43	40	35	38	41

Figure 4: Predicted Notes using built-in LSTM Module in PyTorch.

5 Experiments

PyTorch is a popular Deep Learning library. It allows implementation of existing neural network types, such as RNN, LSTM, MLP, and also provides the flexibility to implement custom Neural network architecture by explicitly defining operations in both forward and backward passes.

Initially, the implementation of the network as described in Method was sequentially implemented in PyTorch, using pre-built LSTM modules as illustrated in Figure 3. The underlying functionality of these modules is to make 1 complete pass over the whole input data (input subsets), based on the number of Epochs. The output of the first layer was then fed into the second LSTM layer, after certain inputs were randomly set to 0 according to the probability defined in the dropout layer. However, this resulted in very poor predictive qualities. Assuming the input data was divided into 30 subsets, each sequence of length 10 timesteps, in 1 forward pass, the first LSTM layer would predict the output values based on all 30 subsets. Therefore, the second LSTM layer was essentially computing the forward pass only on the single predicted vector at timestep t. Since the second LSTM layer was responsible for feeding output to the final layer, the theoretical memory of the network was severely limited, leading to high variance in the predictions as shown in Figure 4.

The values indicate velocity of a certain pitch at each timestep. The converted track was subsequently erratic, with sharp jumps in the same note leading to highly fluctuating audio.

The network was modified, and a custom LSTM network was implemented as illustrated in Figure 5. PyTorch offers modules called LSTMCells, which allow the user to define how the hidden and cell state from each layer is to be utilized. The modified network would now take 1 data point of the original 30 subsets of the input data, feed it to the first LSTM layer, and produce the output to be fed to the second LSTM layer manually. Hence, a single forward pass now encompassed 30 iterations of the input data subset, across both LSTM layers. This enabled the network to retain much larger instances of memory and improved the quality of the output.

Since the input data was sanitized by removing silence periods, the input vectors only consisted of non

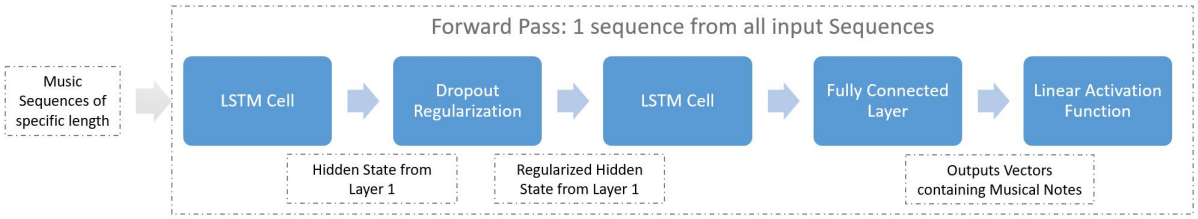

Figure 5: Neural Network architecture using a custom implementation of LSTMCell Modules in PyTorch

40	40	40	40	40	40	40	40	34	34	34	34	34	34	34

Figure 6: Predicted Notes using custom LSTMCells in PyTorch.

zero velocity values for each pitch, as opposed to larger vector size consisting of zero values. Therefore, the sequence size was set to a small value, to ensure any changes in the musical notes would be appropriately captured. Different values were experimented with, including 4,8,12 and 24; whilst 24 led to the fastest convergence, the error rate was slightly higher than 4 which offered the slowest convergence. Hence, a moderate value of 12 was eventually chosen. Similarly, a step size value of 6 was chosen, as too large a step size would overshoot musical notes, and not properly learn the musical structure of the track, while a small step size would repeat notes that the network has already trained on, leading to overfitting.

Dropout probability was set at 0.2, as high values tended to overwrite the output of the first layers with zeros frequently, generating longer periods of silence. The Epoch size was set a very high value, and stopping criterion was used once value of the error was below a certain acceptable threshold. Default value for learning rate of Adam algorithm (1e-3) was used as it lead to best possible convergence.

6 Results

As described in the preceding section, the modified LSTM implementation led to improvement in predictions. Figure 6 shows the output of the final structure.

It is evident that the strong fluctuations from the first implementation have been rectified. However, the important thing to observe is that there are no silence periods anymore, since the network was trained on sanitized data. When the output was transformed back into MIDI files, the audio was more coherent in terms of stability but did not exhibit structural integrity expected from a musical composition. One possible explanation is that the input data contained notes only from a particular type of instrument, which was predicted repeatedly by the LSTM after full round of training. Without incorporating silence periods, the output sounds becomes disorganized.

Moreover, MIDI files contain events like SetTempo, which dictate the value of BPM of beats per minute, for a particular musical note (pitch), at a time interval. These events were not included in the data set, and therefore reduce the likelihood of retaining structural integrity.

The difference in the size of MIDI features vector containing silence and non-silence periods was approximately in the order of 10. It turned out to be impractical to train the neural network, with large sequence sizes, on the former data sets, as it would eventually lead to memory overflows (encountered on a Microsoft Azure VM with 32 GB RAM as well). Therefore, it was not possible to conduct a Turing Test since the quality of the output did not resemble a musical composition. Instead, a comparison of both outputs was manually conducted with different test subjects, and the unanimous opinion was that the latter method achieved a higher stability while still being unable to resemble a composed audio track.

7 Discussion

Training neural networks on audio samples require intensive computing resources, since even a small audio composition contains hundreds and thousands of timestamps. Huang et al. trained a similar 2 layered LSTM network, and asked the participants in a survey to rate the quality of the generated music. They reported significant results in terms of how believable the music sounded to the user [9]. It is though to be mentioned that the researchers used a character formatted input, as opposed to audio samples. This can potentially reduce the amount of data points required to train the network significantly, while simultaneously improving the quality of the output.

An important finding was that MIDI files contain data from multiple instruments which have the same pitch range, and therefore, the various tracks cannot be flattened into one single data stream without incurring velocity overlaps. An improvement over the current model can be training the network on audio tracks using formats that encompass sound spectrograms, such as mp3 or wav, as spectrograms have frequency values that incorporate all instruments.

8 Conclusions

Using digitized audio samples did not offer much improvements over transcribed audio data containing musical characters. It is worth noting that as a future hypothesis, cascading of neural networks that individually learn and track changes in tempo, notes from various instruments, and silence periods would theoretically lead to better results in terms of musical integrity and composition. Since the output of the network generated various off-key musical events throughout the prediction, it should be considered that the quality of the network alone cannot determine musical integrity, without inclusion of post processing algorithms that remove such anomalies, leading to smoother transitions between different notes.

References

[1] E. R. Miranda, *Readings in music and artificial intelligence*. Routledge, 2013, vol. 20.

[2] L. A. Hiller and L. M. Isaacson, "Experimental music: composition with an electronic computer," 1959.

[3] C.-H. Liu and C.-K. Ting, "Computational intelligence in music composition: A survey," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 1, pp. 2–15, 2017.

[4] H. S. Howe, "Composing by computer," *Computers and the Humanities*, vol. 9, no. 6, pp. 281–290, 1975.

[5] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning," *arXiv preprint arXiv:1604.08723*, 2016.

[6] S. Shuvaev, H. Giaffar, and A. A. Koulakov, "Representations of sound in deep learning of audio features from music," *arXiv preprint arXiv:1712.02898*, 2017.

[7] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[8] I. Liu, B. Ramakrishnan *et al.*, "Bach in 2014: Music composition with recurrent neural network," *arXiv preprint arXiv:1412.3191*, 2014.

[9] A. Huang and R. Wu, "Deep learning for music," *arXiv preprint arXiv:1606.04930*, 2016.

[10] MIDI association. [Online]. Available: https://www.midi.org/

[11] J. Weel, B. O. K. Intelligentie, and E. Gavves, "Robomozart: Generating music using lstm networks trained per-tick on a midi collection with short music segments as input." 2016.

[12] G. Hall, "Python midi: A library for midi in python." https://github.com/vishnubob/python-midi, 2016.

Roles of the authors

For this project, the authors divided the responsibilities in two: Mohammad Omar Nasir was in charge of the programming while Laura Deleuze was in charge of the report. In other words, each one mainly lead one part for efficiency purposes. Yet, both equally participated in all tasks: the datasets, resources, algorithms and methods were identified, defined, developed and refined together.