Music generator with Recurrent Neural Networks

Claudia Ivett Núñez Martínez A01700762

I. INTRODUCTION

Creating music following specific harmonic structure is not an easy task for humans since we need to map continuous and simultaneous sounds into short phrases that finally creates a different communication channel. The ability to predict upcoming structured events based on long-term knowledge and contextual priors is a fundamental principle of human cognition [1].

What about letting this task to a computer? This is an active research subject in neural networks. Many approaches and solution proposals include harmonic analysis, polyphonic or monophonic music generation.

Recurrent Neural Networks with Long Short Term Memory can generate music based on previous sequence-to-sequence learning. This type of neural network is particularly useful for speech recognition, natural language and other analysis types with data constantly changing.

II. BACKGROUND

Recurrent Neural Network (RNN)

“Recurrent neural networks constitute a broad class of machines with dynamic state; that is, they have state whose evolution depends both on the input to the system and on the current state” [2]. It is mainly used for Natural Language Processing since it is necessary to “read” and remember previous words or states in order to predict the next one.

Sequence-to-sequence learning works for RRNs. This approach can be seen as mapping a sequence of words representing the question to the corresponding sequence of words equivalent to the answer for that question [3].

For this project the input is a set of sequences, getting one sequence as output as shown in Figure 1, where the input layer is represented by red boxes, hidden layer is shown in green boxes and the blue box is the expected output.

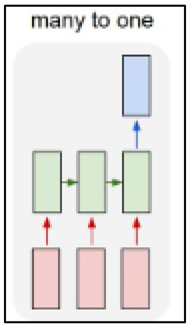

Figure 1

One of the most important characteristics of RNN is that they take as their input not just the current sequence, but also previous outputs. This means that each output is influenced by weights but also by a hidden state vector containing prior information . We can infer then that the same input can produce a different output depending on previous inputs in the series.

Parameter sharing is another fundamental element for RNN. Same weights will be applied over every input since a RNN does not receive a single input as we have seen in other neural networks, it gets multiple input sequences and the same task is performed over each one.

With this information, now we can go through the construction of a RNN. Having an input x_t, an output y_t and hidden state h_t where t represents time, we will end up with the following transition function f_t and the output function f_o as shown below:

$$h_t = f_t(x_t, h_{t-1})$$

$$y_t = f_o(h, x_t)$$

Long Short Term Memory Network

Sometimes the information required from the past to predict the next movement (next word in a sentence or next note in a melody) can be obtained from the last two or three movements, but in some other particular cases it is necessary to go further back to get more context and improve the prediction output. So it's entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information [5]. Solving this problem is the main purpose of Long Short Term Memory Networks (LSTMs). It is a special kind of RNN able to learn long term dependences.

Important elements for music analysis

The main elements for this approach are musical notes, which represent each sound associated with a specific pitch (frequency), duration and offset. Taken this as the base sound form, we will find also chords, which are a set of notes played at the same time, and rest times that implies a temporal sound pause.

Time step is the unit time that determines the shortest possible duration for a note. Sequence length is the unit for learning and predicting patterns. For this approach a time step of 0.25 seconds was selected and 8 notes for the sequence length.

III. MODEL

Neural Network architecture

Input LSTM network layer receives sequences previously generated by parsing midi files. Since it is the first layer, the input shape has to be specified. In order to ensure having an output from each unrolled LSTM cell through training execution, the return_sequences parameter was set to true. Diagram at Figures 2 and 3 illustrates how it enables the comparison at each training step. This configuration is kept through stacked LSTM layers.

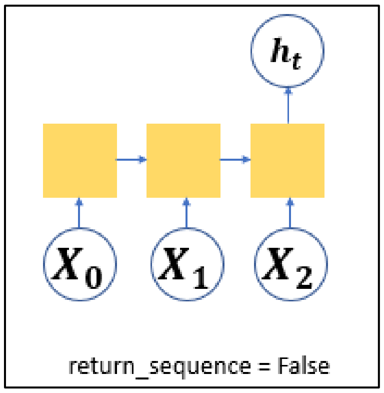

Figure 2

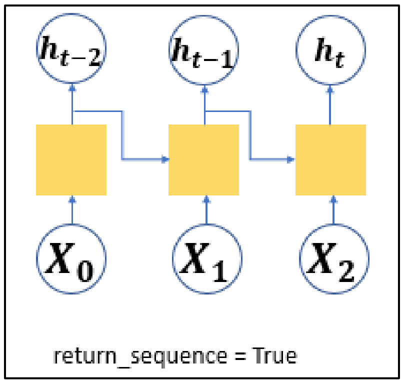

Figure 3

Dropout layers are used for preventing overfitting. This regularization technique consists in ignoring (setting to 0) a fraction of input nodes. Even though it increases the number of iterations needed to converge, the training time for each epoch was reduced. A more detailed description will be shown in Implementation section.

Cross entropy was selected for loss calculation since for each output belongs to a single instance from our vocabulary or classes (notes set).

IV. IMPLEMENTATION

Input data

Dataset contains 262 music files in midi format. They contain piano pieces from Romanticism and Classism periods, including composers like Edvard Grieg, Franz List, Schubert, Chopin. It was obtained from a repository hosted at *http://www.piano-midi.de/midicoll.htm* [6].

Music21 is a Python library useful for processing midi files. It generates steam, note and chord objects that then are parsed into string format so that we can get a notes vocabulary, where a unique note will be conformed by its name (C, D, E, F, G, A, B) and duration (considering the time step is 0.25 seconds).

Each note has an offset property that indicates the time in which it starts. The offset from current processing note and the previous one is required to calculate rest times. Figure 4 illustrates offset and timestep for notes.

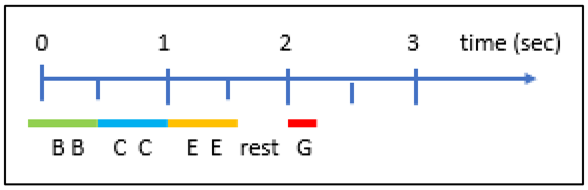

Figure 4

Sequences are created using the notes vocabulary and the notes file that contains every note parsed during data input processing.

Figure 5 illustrates how input sequences and the corresponding output are created. Before performing this operation, a mapping function is used to create a dictionary to map notes pitches to integers values.

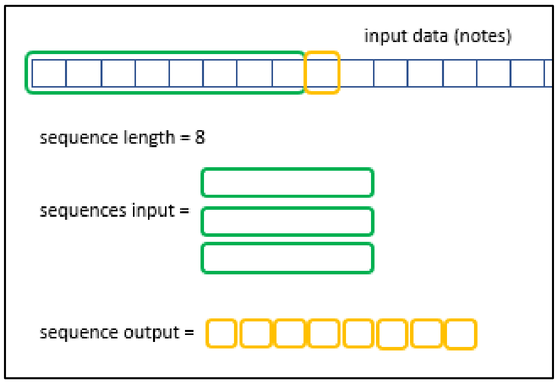

Figure 5

Finally, vectorization, normalization and one-hot encoding techniques are performed in order to have an input compatible with what our network expects.

Training

The callback function ModelCheckpoint from Keras was used for tracking the model through training process by saving the current model state at the end of each epoch. Even that I trained the model with 200 epochs, I can use any model that was saved after each epoch in case the best metrics are reached before running 200 epochs.

Generator

Creating music requires same setup configuration used for training environment. The difference for this phase is that now the weights from the trained model are loaded.

The input sequences for music generation are obtained from the notes file previously parsed and used for training. This time a random sequence index is selected as the starting point for prediction.
Since the hidden state will load sequences in an unknown order, we can get different results every time.

V. RESULTS

The RNN achieves loss of 0.0244 after 90 epochs. Following graph shows the accuracy and loss through training process.

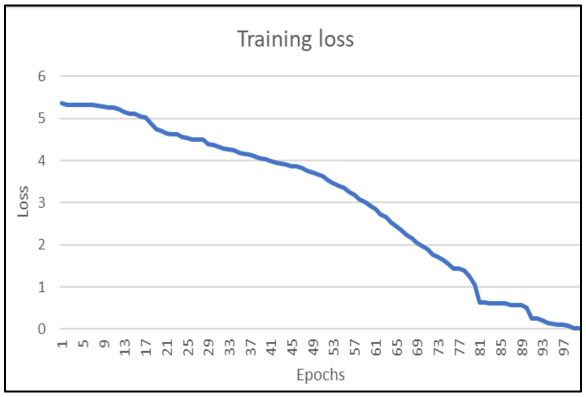

Figure 7 and Figure 8 shows the resulting music sheets from model predictions. We can see here the importance of rest notes, duration and a defined time step since those elements increase the complexity of the music generated.

We can notice that the model returns different results even that the whole set of sequences used was the same but feeding random starting point.

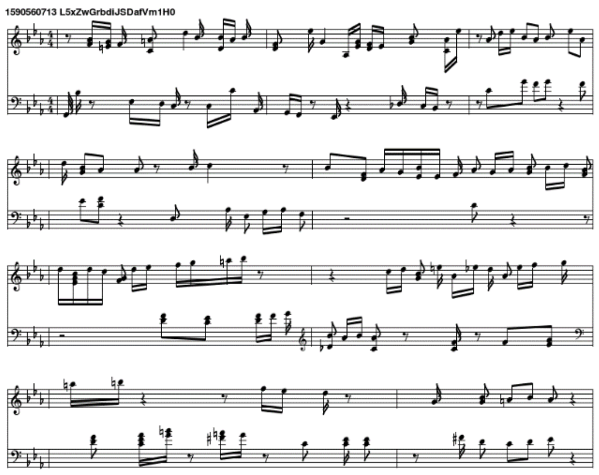

Figure 7

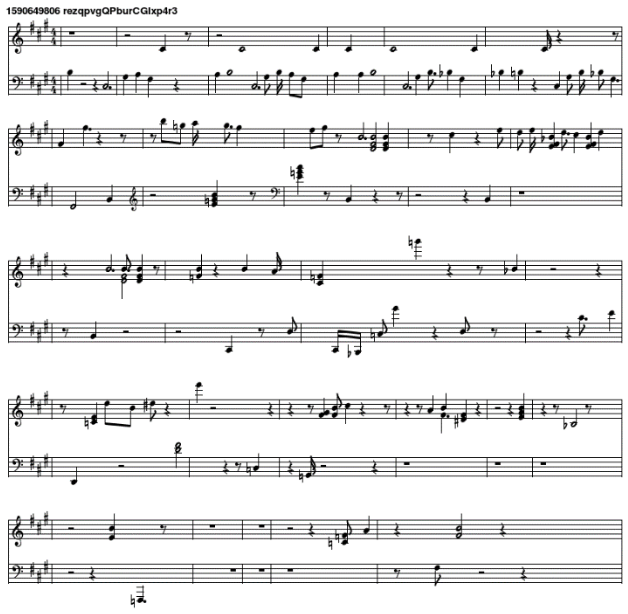

Figure 8

The most important thing for implementing a solution in an extensive field as Neural Network, is understanding what is happening behind of apparently simple lines of code that is probably supporting a whole math foundation for our algorithms. This allows us to design a more accurate and efficient architecture and keep improving it based on results interpretation

VI. FUTHER WORK

Implement learning in a deeper scope including different harmonic structures.

Identify the intention or sentiments from music. Also, creating music that reflects a specific feeling.

Create music from image patterns.

VII. REFERENCES

[1] https://www.sciencedirect.com/science/article/abs/pii/S1053811916304116?via%3Dihub

[2] https://arxiv.org/pdf/1410.5401v2.pdf

[3] https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[4] https://arxiv.org/pdf/1312.6026.pdf

[5] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] http://www.piano-midi.de/midicoll.htm

[7] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Implementation was based on work from:

[8] Sigurður Skúli https://github.com/Skuldur/Classical-Piano-Composer

[9] Tyler Doll https://github.com/tylerdoll/music-generator