Chord Generation for input melody in MIDI format using LSTM and GRU with attention mechanism

Antonia Mouawad and Fady Baly
Department of Computer Science
University of Toronto
1004503487
antonia@cs.toronto.edu

Abstract

Chord composition based on a monophonic melody is a tedious task for most of the music amateurs as it requires a decent understanding of music theory and chord progression. In this paper we present different methods for generating musical chords that correspond to an inputted melody in Musical Instrument Digital Interface (MIDI) format. For that, we preprocess the monophonic melody and normalize it in order to ensure the uniformity of the note duration, the time signature and the key amongst all the dataset. We train the Recurrent Neural Networks (RNNs) on the preprocessed data with long short-term memory (LSTM) cells on one hand and Gated Recurrent Unit (GRU) on the other. For further testing, we add an attention layer for each network configuration, resulting with a total of four different models. The quantitative comparative analysis shows GRU based RNN outperformed all the other networks (67.41% accuracy). Knowing that music can be subjective, and many chord generations might be satisfactory, a user qualitative study was also conducted which resulted in the same conclusion.

1 Introduction

Music is a scientific art. It is a science in its nature, vibrations and sounds, and its structure, the rhythm, the beat, and the music theory. It is an art, an expression of oneself, and this expression can take a variety of forms reflected for instance in the different variations and genre covers for the same song. This amalgam of science and art makes music generation interesting and challenging to this day. The most basic splitting of any song is into the monophonic melody and its corresponding chord progression. While the melody serves as the main focus of the music, the harmony enriches it and adds to it another dimension. Composing music melody is achievable without a former background in musical studies and theory. This is partly due to the fact that the human brain focuses on the melody and keeps the harmony as a background sound [1]. Consequently, with a bit of talent, music amateurs might be able to generate melodies, however accompanying them with matching chords can be difficult. We try to overcome this limitation by exploring the possibility of creating an autonomous and automatic chord generation system.

As the RNNs present an elegant way to deal with sequential and temporal data [2], since every neuron can keep information about the previous input, we build a model to predict chord progressions using two extensions of RNNs, LSTMs and GRUs. We then compare the results between those two methods on one hand and between them our baseline model using CNNs, which do not take the overall music progression into account on the other hand. We use 63 piano MIDI songs from www.hooktheory.com, although it has been challenging to collect it, as opposed to using ABC format or lead sheets. The reason behind this decision is that we wanted our system to be as realistically useful for novice pianists and musicians as possible. With nowadays recording one's melody in MIDI format a very easy process, facilitated by many free user-friendly softwares [3] as opposed to writing the melody on lead sheets, which necessitates the ability to read and write music in the first place and requires a prior knowledge in music dynamics.

2 Related Work and Background information

2.1 LSTM and GRU based RNN music composition projects

Although computer scientists have gone a long journey in music analytics, most of the projects aimed at creating music from scratch, in other words creating the melody and the harmony at the same time, instead of building one on top of an inputted other. For instance, MelodyRNN from the Magenta project by Google Brain [4], tried to create compelling music after training the LSTM-based RNN on the NSynth data, in wav form. Similarly for the project Bach in 2014 [5], LSTM networks have been shown to be effective in recreating music after properly learning its structure and the characteristics. However in an empirical study on GRU-based RNNs trained on polyphonic sequential MIDI music [6], GRUs were found to outperform LSTMs in efficiency and accuracy. DeepBach, an RNN trained model by Sony CSL, allows injecting user-defined inputs such as rhythm, a few beginning notes and starting chords [7]. While all these project applied RNNs on temporal data, none of them focused on modeling the melody and the chords as two separate entities, and finding one based on the other. In this sense, our project is innovative as we propose a data engineering method to preprocess the MIDI music, normalize it, separate it into chord and melody, and then feed it to the network where, the melody progression is the input, and the corresponding chord progression is the output, using LSTM and GRU based RNNs. We then extend our experimentation by adding an attention layer that precedes the output layer both networks, described in section 4.4.

2.2 Data representation

Another dimension to be carefully considered was the representation of the musical content, in other words the format of the input and the output. In a study on algorithmic music generation [8], musical information was extracted from raw audio signals and their derivatives (waveforms, spectrograms, etc.). Although this representation would have been useful in chord generations for novice pianists, music collection is susceptible to noise and might not be accurately translated to its mathematical representation. Data lead sheets are another widely available representation format for popular genres of music. They which provide both an accurate description of tempo and lengths of notes, and separate naturally the melody (scores) from the harmony (letter chords written above the melody)[9]. The main reason that led us to disregard this rich and concise representation is because the novice musicians would have to write their melodies using proper scores, and guessing the keys, the tempo, the time signature and other musical characteristics, which is almost impossible for amateurs who play by ear.

Most current systems and experiments on music analytics focus on symbolic representations [10]. MIDI is a technical standard protocol for a digital interface of instruments or synthesizers. A MIDI message carries information about the notes played (pitch, velocity, time on and off) and the channels on which they are played [11]. Another commonly used symbolic representation is the piano roll, and it is the main way that Synthesia and piano tutorial softwares use for visualization [12]. The main difference between MIDI and piano rolls is that the former describes many single note messages, whereas the latter describes all played notes at a specific instant. Further details will be elaborated in the next section, however it is important to understand, as Huang and Hu deduced in their experiment on the difference between MIDI and piano rolls [13] that to each its advantages and drawbacks. Inspired by an idea found on the internet proposed by Lackner, we decided to benefit from the best of both the worlds, by changing the MIDI files to piano rolls in an innovative and reversible way, feed the latter to the network, and reverse the piano roll output to MIDI.

3 General System Description

3.1 Data collection

Finding the right dataset for our project was the first challenge. Music informatics is known to be a data-poor field especially due to copyrights issues. For instance, Wikifonia.org, the most common public lead sheet repository stopped service as of 2013 for the same aforementioned reason. We subscribed for the services of Hooktheory.com, from where we downloaded 63 songs in MIDI format, 53 for training and 10 for testing. The songs are all "normalized" by shifting them back to the key "C", as proposed by [14, 15]. Transposing from a key to another is done by simply adding/subtracting a constant to every pitch in the midi format, this constant being the musical "distance" between the two keys (see table 1). Therefore, the dataset becomes more organized, and the chords are significantly reduced to those related to key C, without any loss of musical information.

Sample of key C chord letters	Corresponding MIDI Values	Sample of key C# chord letters	Corresponding MIDI Values
Cmaj	0, 4, 7	C#maj	1, 5, 8
Dmin	2, 5, 9	D#min	3, 6, 10

Figure 1 Orange section of the table corresponds to C key chords, blue section corresponds to C# key chords. C and C# are separated by 1 musical tone, consequently notice how the according MIDI Values are just separated by 1 (0, 4, 7 versus 1, 5, 8 for instance)

3.2 Preprocessing

3.2.1 Data Cleaning

The first step for preprocessing the data was to split it into two parts: the melody (taken from channel 1 of the midi files), and the chords, from channel 2. Following the data split, we decided to limit the range of the pitches from 60 to 71 that is to cover one octave, as shown in the red rectangle below. Anything outside this range is taken back to it by adding/subtracting multiples of 12, with 12 being the distance between two same notes separated by one octave (for example the distance between the C within the red square and the C at the right of it), corresponding to shifting by multiples of octaves. This will result with the same note, but played with a deeper or a higher octave, thus the number of features (pitches) is smartly reduced significantly.

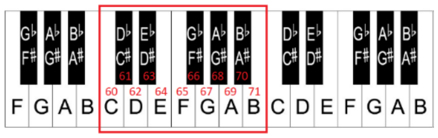


Figure 2 Piano keys with corresponding MIDI number in red, we transposed any pitch outside the red square to the equivalent pitch inside it, thus reducing the number of features to 12 (C, C#, D, D#, E, F, F#, G, G#, A, A#, B)

3.2.1 From MIDI to piano roll to network input

To give a clear understanding of this important step, we will take this chord from one of the pieces, and show its corresponding MIDI representation, and explain how we changed it into piano roll.

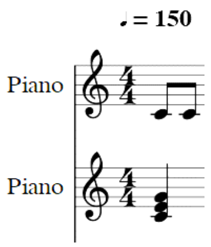


Figure 3 Melody 1st line, chord 2nd line

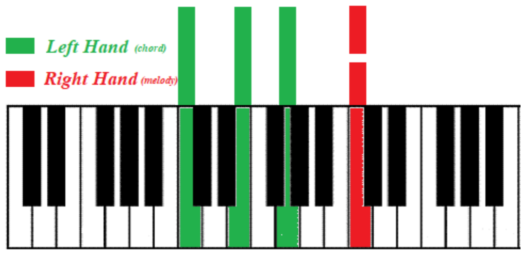
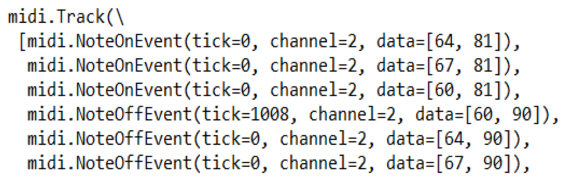
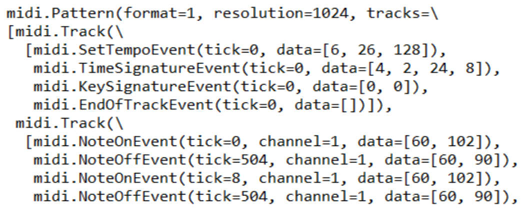


Figure 4 Piano Roll Representation



```
midi.Track(\
 [midi.NoteOnEvent(tick=0, channel=2, data=[64, 81]),
  midi.NoteOnEvent(tick=0, channel=2, data=[67, 81]),
  midi.NoteOnEvent(tick=0, channel=2, data=[60, 81]),
  midi.NoteOffEvent(tick=1008, channel=2, data=[60, 90]),
  midi.NoteOffEvent(tick=0, channel=2, data=[64, 90]),
  midi.NoteOffEvent(tick=0, channel=2, data=[67, 90]),
```

Figure 5 MIDI representation of the left hand section

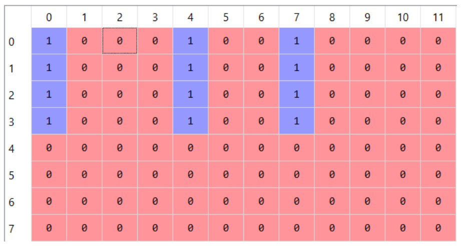


```
midi.Pattern(format=1, resolution=1024, tracks=\
[midi.Track(\
  [midi.SetTempoEvent(tick=0, data=[6, 26, 128]),
   midi.TimeSignatureEvent(tick=0, data=[4, 2, 24, 8]),
   midi.KeySignatureEvent(tick=0, data=[0, 0]),
   midi.EndOfTrackEvent(tick=0, data=[])]),
 midi.Track(\
  [midi.NoteOnEvent(tick=0, channel=1, data=[60, 102]),
   midi.NoteOffEvent(tick=504, channel=1, data=[60, 90]),
   midi.NoteOnEvent(tick=8, channel=1, data=[60, 102]),
   midi.NoteOffEvent(tick=504, channel=1, data=[60, 90]),
```

Figure 6 MIDI representation of the right hand section

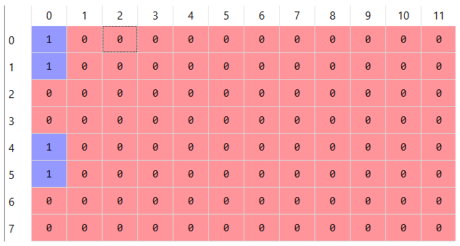
The resolution of our files is 1024, as shown in figure 6. This means that every quarter note (the timing for the chord) is represented by 1024 ticks. This very high resolution is not needed, as we will explain later. The first value in data indicates the pitch being played (a number between 60 and 71), and the second represents the volume on which it is played. The NoteOnEvent, and NoteOffEvent indicate a note being pressed and released respectively, at time = tick with respect to the previous event.

Obviously, this representation is not suitable for the networks to be trained on, for this reason we changed the midi representation to one accessible and acceptable by the RNNs, a representation similar to the one shown in figure 4, piano rolls using binary numbers. The first key change was dropping this very unnecessarily high resolution. We decided to take the 1/16th note to be the fastest representable pitch time instead of 1/1024th. However, as mentioned previously in [13], piano rolls are unable to distinguish between two short consecutive notes and one long note. For instance the two 1/16th C in the melody in figure 3 would be read exactly the same as of having one 1/8th C pitch. For this, we append the time of the pitch with an equal number of 0s. Consequently, representing a 1/16th note (shortest duration possible in our system) requires 2 rows: one row containing ‘1’ at the location of the notes to be pressed, and one all zero row to indicate turning off the notes. Similarly, to represent 1/8th note, we need two rows indicating ‘1’ in the column of the corresponding pitches, and two rows of zero before moving on to the next input. Below is shown the equivalent of figures 3, 4, 5, 6 (one beat, i.e ¼ note).



	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	1	0	0	1	0	0	0	0
1	1	0	0	0	1	0	0	1	0	0	0	0
2	1	0	0	0	1	0	0	1	0	0	0	0
3	1	0	0	0	1	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7 Roll representation of the left hand section



	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8 Roll representation of the right hand section

The last step of the preprocessing is to set the sequence length to be inputted to the network. The sequence length was set to four measures, which is 16 beats or 128 ticks, as the chords change every other beat. We input the first t as the sequence for the RNN, with t = sequence length, and the corresponding target is the chord at t + 1, this results in 13568 datapoints

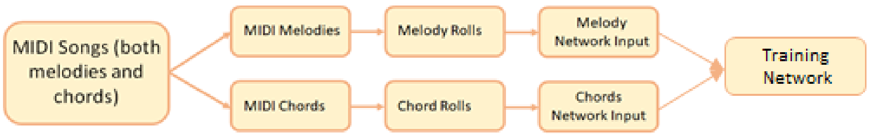


Figure 7 Steps to train the network

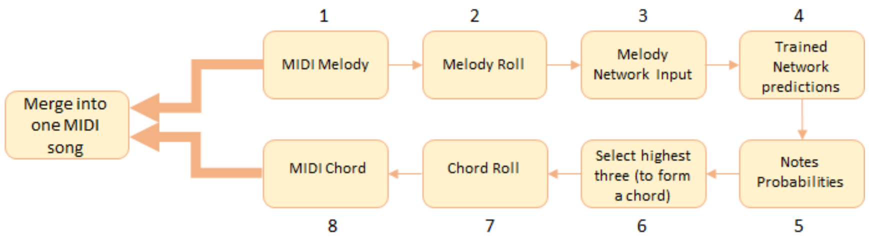


Figure 8 Steps for Chord generation and music merging

4 Neural Networks

4.1 Recurrent Neural Networks

In a Recurrent Neural Networks, the output of a neuron is fed back into its own input (closed loop) and also into the input other neurons in the following time step. This allows to keep a memory of the information of previous steps, thus RNN are able to model sequential and time dependent data. However, the major drawback in RNN is the vanishing or exploding gradient, which makes learning long-term dependencies difficult [16]

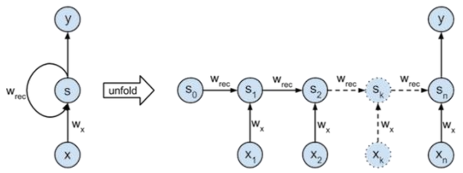


Figure 9 Simple Recurrent Neural Network [17]

4.2 LSTM

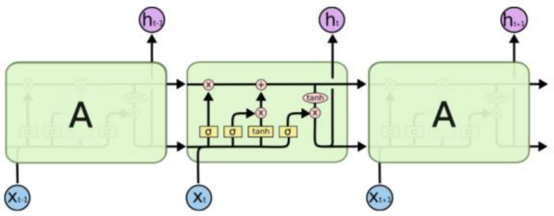
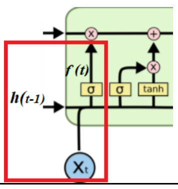
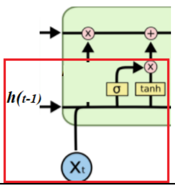
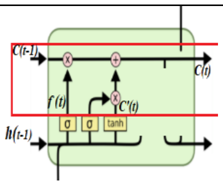
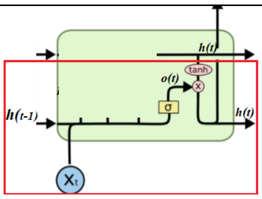


Figure 10 Repeating module in an LSTM [18]

The repeating module in the LSTM contains four interacting layers, making long term dependency modeling the default option in this network. We will briefly explain the role and the mathematical equation of each.

Step			 C(t-1) C(t) f (t) C'(t) h(t-1) σ σ tanh	
Mathematical equation	$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$	$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$ $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$	$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$	$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$ $h_t = o_t * \tanh\left(C_t\right)$
Explanation	The sigmoid "forget gate layer" looks at the previous hidden state and the current input and outputs a value between 0 and 1 for every cell state Ct-1(0 represents" completely forget this"	This step decides which values will be stored in the cell state. The sigmoid is also a forget gate layer, and the tanh creates possible candidates Ĉt	Here the new cell Ct is then updated according to the calculations from the previous steps.	The final output is a filtered version of the cell state. Tanh is used to obtain values between -1 and 1 and multiplied by the output of the third sigmoid gate, to keep only the needed values.

4.3 GRU

The idea of gated recurrent units is relatively new: it was proposed in 2014 and its advantages and drawbacks have not yet been well explored. Although, as mentioned before [6] underlines that there is no clear winner between LSTM and GRUs, GRUs have less parameters and usually outperform LSTMs on smaller datasets as they can generalize better and can be trained faster. It is no doubt that LSTMs and GRUs are very similar in function and equations:

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r)$$
$$h = tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

Figure 11 Equations for GRU [19]

The key difference between a GRU and an LSTM is that the former has two gates (sigmoid functions) as opposed to the latter which has three. Moreover, a GRU doesn't possess the state cell C_{t-1}, instead it only preserves the exposed hidden state. Last but not least, the tanh function is not applied at the output (no second non linearity after the sigmoid).

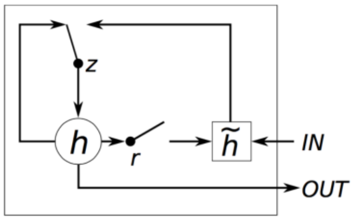


Figure 12 GRU gating [6]

4.4 Attention Mechanism

Attention Mechanisms in Neural Networks were loosely inspired by the way we perceive an image, focusing at the beginning at the most important part, or the high resolution section, and blurring out the less important part, or the low resolution section [20, 21]. The main advantage of an attention mechanism instead of encoding the full melody roll into a fixed sized vector, we can allow the decoder to *pay attention* to the various parts of the training data at every step of the chord generation following the attention weight α_j. This attention weight can be thought of as the probability that the decoder selects the jth pitch out of all the other possibilities.

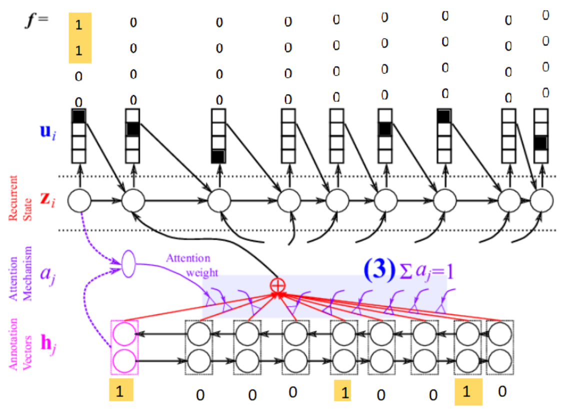


Figure 13 Our Network with attention for music generation, inspired by NLP network proposed in [22]

5 Experiments and network implementation

5.1 Data Analysis

Finding a good RNN model requires many iterations and parameter tuning. One major challenge was coping with the imbalanced data, especially due the uniformity of the key (preprocessing). As shown in the histogram below, most of the chords in the training set were C Major which resulted in the network mostly if not always predicting C Major for the test melodies.

Most Frequent Chords	**Value in Dictionary**
C Major	1
F Major	6
a minor	23

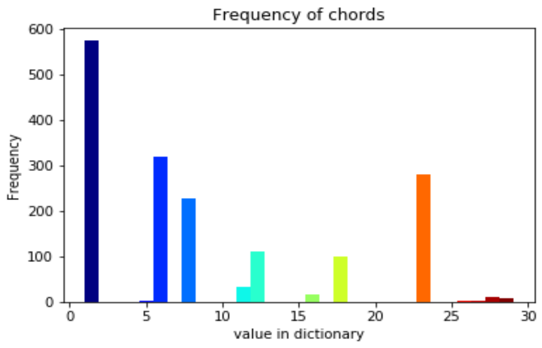
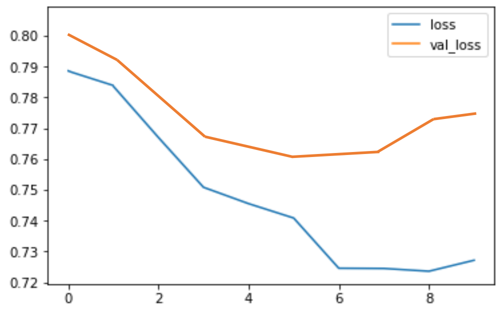


Figure 14 Most frequent chords in the dataset

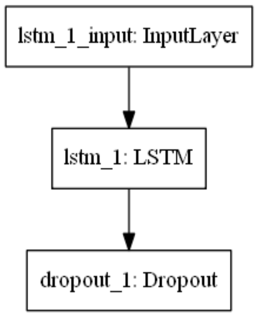
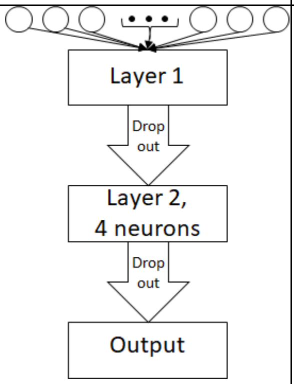
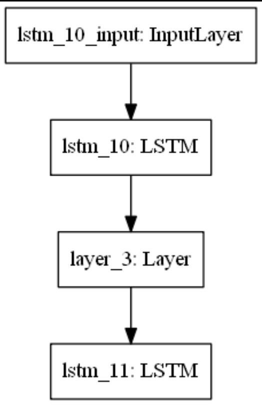
The loss function we used was the "binary_crossentropy" from Keras, as the output is simply 1's for the pitches to be played and 0 for the others. At the beginning the most frequent class, C major (or 1 on the pitches C, E, G and zero elsewhere) is dominating the loss - so network is learning to predict mostly this class for every example. Hence reducing from the majority class was necessary, paralleled with an upsampling of the minority class, to ensure a uniform-like distribution.

5.2 Loss and Epochs

Another challenge we faced was the overfitting of the networks to the relatively small dataset. In order to solve this issue, two major steps were taken. First, a dropout rate of 0.25 was added as suggested by [23], which drops randomly units and their connections from the network during training to avoid the units from co-adapting too much. Secondly, a subset of the training set (Validation set) was kept aside and used to test the loss after various epochs to determine the best number of epochs to stop at before overfitting (at the point where the validation loss increases while the training loss is still decreasing)

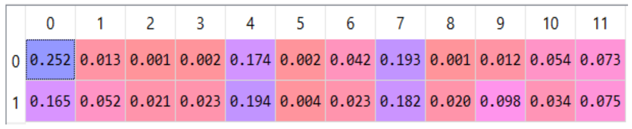


5.3 Network architectures

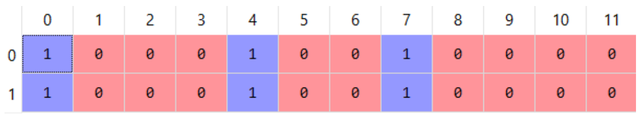
Name	LSTM	GRU	(GRU,LSTM)+attention
Network architecture	 lstm_1_input: InputLayer lstm_1: LSTM dropout_1: Dropout tanh activation, sigmoid at output	 Layer 1 Drop out Layer 2, 4 neurons Drop out Output	 lstm_10_input: InputLayer lstm_10: LSTM layer_3: Layer lstm_11: LSTM
Rationale and discussion	The LSTM based RNN was overfitting way too easily on the data. This is actually an expected result, as this network performs best with larger datasets. After iterating over different architectures, the one that achieved the best sounding music was a shallow network.	Similar structure to LSTM, however it could tolerate more 1 more layer composed of 4 neurons without overfitting. Keep in mind that dataset is not very large (13568 instances).	With the attention (here called layer_3) the networks were both able to be a bit deeper, the performance was better for both. Similar architecture for GRU

5.4 Output and Post-processing

The output of the network is the probability of every pitch according to the inputted melody sequence. In order to obtain a triad chord (chord composed of three notes), we find the highest three probability in every output sequence, then change the output back to MIDI as explained earlier.



	0	1	2	3	4	5	6	7	8	9	10	11
0	0.252	0.013	0.001	0.002	0.174	0.002	0.042	0.193	0.001	0.012	0.054	0.073
1	0.165	0.052	0.021	0.023	0.194	0.004	0.023	0.182	0.020	0.098	0.034	0.075



	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	1	0	0	1	0	0	0	0
1	1	0	0	0	1	0	0	1	0	0	0	0

6 Testing

6.1 Accuracy

The accuracy by definition is the ratio of the correctly classified to the total. The accuracy of our system was as follows:

Network	LSTM	LSTM + attention	GRU	GRU + attention
Accuracy	61.2%	62.51%	66.23%	67.41%

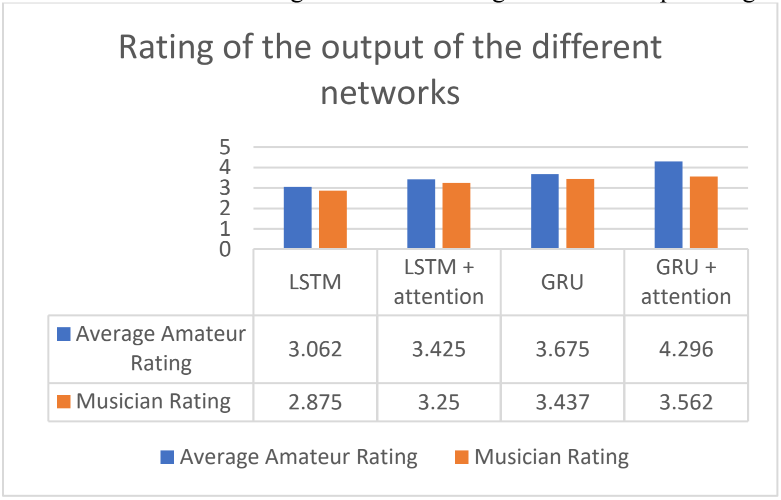
Although the accuracy of the system can be useful for a general overview of the performance, it is not totally descriptive for two reasons.

1- The imbalanced data (even after correction) tends to push the network to always predict the most frequent chord (further improvement should be made here)

2- Many different chords can sound good and be placed on top of the melody (and theoretically correct). For instance the GRU and the GRU with attention almost have the same accuracy, however the latter sounds better than the former.

6.2 User study

For this task, we asked 5 people, 4 music amateurs and 1 musician to evaluate the outputted music according to the following scheme: For every predicted chord they must rate on a scale from 1 to 5 how musically pleasant this chord sounds. As a reminder, every song in our dataset is composed of 8 measures or bars, and in each 2 chords are generated resulting in 16 chords per song.



	LSTM	LSTM + attention	GRU	GRU + attention
🟦 Average Amateur Rating	3.062	3.425	3.675	4.296
🟧 Musician Rating	2.875	3.25	3.437	3.562

The musician quoted: “I could tell when that your system is detecting some good chord progressions, however it still lacks the small tweaks that only a trained musical ear can detect”.

6.3 Links

Code: https://github.com/AntoniaMouawad/Chord-Generation-using-LSTM-GRU
Sample of generated music: https://youtu.be/DZDoU_zTR3Y

7 Limitations and future work

Although the results seem promising, many steps can be taken for a better performance and a more pleasing music. First of all, on the technical level, instead of predicting separate pitches, which might cause the formation of incorrect chords on the theoretical level, letting the last layer of the network directly predict the chord (for example the first index is C Major chord, the second is A minor chord, etc). However, this necessitates a larger dataset, especially if more genres/keys/chord progressions are covered to avoid sparse matrices. In all cases, increasing the dataset size would definitely enhance the model performance, as it is tightly related to the network training. Furthermore, better data balancing techniques should be investigated to avoid redundancy and constant prediction of the most frequent class. Finally, allowing for a discrimination among genres can significantly improve the predictions as every genre has its different chord progressions.

8 Conclusion

In this project we tried to implement a smart chord generation system to help novice pianists and musician accompany their composed melodies. We suggested a methodology for MIDI preprocessing in order to be ready for training the neural network, we investigated different architectures and cell structures for the network, and outlined the limitations of our work.
Chord generation is a tedious task, and tuning the networks was harder than expected, but the outcome was promising, especially for the GRU based network with attention mechanism. Although this project was not able to fully succeed in generating a perfect harmony match for the melody, it proved that neural networks nowadays are able to complete tasks that were previously thought impossible.

9 Acknowledgments

I would like to thank all the professors who taught me this interesting yet challenging course. I am also very thankful for my boyfriend, a very talented pianist, who was by my side suggesting ideas during the project, and to all the music amateurs who agreed on testing and rating the generated music.

References

[1] Brown, S., Martinez, M. J., Hodges, D. A., Fox, P. T., & Parsons, L. M. (2004). *The song system of the human brain*. Cognitive Brain Research, 20(3), 363-375.

[2] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006, June). *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks.* In Proceedings of the 23rd international conference on Machine learning (pp. 369-376). ACM.

[3] *The 8 Best MIDI Keyboard Controllers for Home Recording*. (2017, October 30). Retrieved December 5, 2017, from https://ehomerecordingstudio.com/midi-controllers/

[4] Lou, Q. (2013) *Music Generation Using Neural Networks*.

[5] Liu, I., & Ramakrishnan, B. (2014). *Bach in 2014: Music composition with recurrent neural network.* arXiv preprint arXiv:1412.3191.

[6] Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). *An empirical exploration of recurrent network architectures.* In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 2342-2350).

[7] Hadjeres, G., & Pachet, F. (2016). *DeepBach: a Steerable Model for Bach chorales generation*. arXiv preprint arXiv:1612.01010.

[8] Sarroff, A. M., & Casey, M. A. (2014). *Musical audio synthesis using autoencoding neural nets*. In ICMC.

[9] Eck, D., & Schmidhuber, J. (2002). *A first look at music composition using lstm recurrent neural networks.* Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 103.

[10] Briot, J. P., Hadjeres, G., & Pachet, F. (2017). *Deep Learning Techniques for Music Generation-A Survey.* arXiv preprint arXiv:1709.01620.

[11] MMA - MIDI Manufacturers Association. (n.d.). Retrieved December 10, 2017, from https://www.midi.org/specifications

[12] Synthesia, Piano for Everyone. (n.d.). Retrieved December 10, 2017, from http://www.synthesiagame.com/

[13] Huang, A., & Wu, R. (2016). *Deep learning for music*. arXiv preprint arXiv:1606.04930.

[14] Lim, H., Rhyu, S., & Lee, K. (2017). *Chord Generation from Symbolic Melody Using BLSTM Networks*. arXiv preprint arXiv:1712.01011.

[15] Mozer, M. C. (1994). *Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing*. Connection Science, 6(2-3), 247-280.

[16] Bengio, Y., Simard, P., & Frasconi, P. (1994). *Learning long-term dependencies with gradient descent is difficult.* IEEE transactions on neural networks, 5(2), 157-166.

[17] Roelants, P. (n.d.). *How to implement a recurrent neural network Part 1*. Retrieved December 17, 2017, from http://peterroelants.github.io/posts/rnn_implementation_part01/

[18] *Understanding LSTM Networks.* (n.d.). Retrieved December 16, 2017, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[19] Britz, D. (2016, January 10). *Implementing a GRU/LSTM RNN with Python and Theano.* Retrieved December 16, 2017, from http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

[20] Britz, D. (2016, April 27). *Attention and Memory in Deep Learning and NLP*. Retrieved December 17, 2017, from http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

[21] Luong, M. T., Pham, H., & Manning, C. D. (2015). *Effective approaches to attention-based neural machine translation*. arXiv preprint arXiv:1508.04025.

[22] *Introduction to Neural Machine Translation*. (2017, April 10). Retrieved December 17, 2017, from https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

[23] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting.* Journal of machine learning research, 15(1), 1929-1958.