POPULAR MUSIC GENERATION WITH DEEP LEARNING TECHNIQUES

https://github.com/Tiff923/Music-Generation.git

Yee Wang Sui, Jessica Davina, Goh Yi Lin Tiffany, Chow Jia Yi

December 2020

1 Problem

There is nothing that makes us more human than our ability to be creative. Yet, in recent years, artificial intelligence has been advancing in the more creative pursuits, particularly music generation. Although there were promising results, much of existing research used classical music to build AI models. Taking this into consideration, we decided to look into building machine learning models that could generate music that appeals to the masses. This project aims to generate popular music based on the patterns learnt in video game songs. We define popular to be quantified by the ratings and reviews of the music pieces given by people.

2 Dataset and Collection

A few decisions were made before sourcing for the datasets of music pieces. Firstly, we decided on single instrument pieces to reduce complexity of the music generation as we were new in this domain. Secondly, instrument of choice was the piano, as it covered both bass and treble clefs, and can be used to substitute entire orchestras. This would allow our models to learn musical information about both the melody and underlying chord progression. Thirdly, the genre was chosen to be video games, as they were usually more elaborate pieces.

As a result, two datasets were used, namely, an existing VGMIDI dataset and another self-sourced popular video game soundtrack dataset. The VGMIDI dataset was sourced from the 2019 paper "Learning to Generate Music with Sentiment" by Ferreia, Lucas N. and Whitehead, Jim. It consisted of 204 MIDI labelled piano pieces of video game soundtracks.

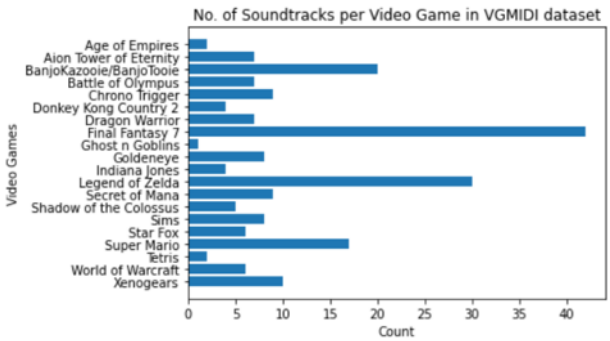

Figure 1: Bar chart of the number of soundtracks per video game in VGMIDI

Another video game soundtrack dataset of 98 popular songs were also collated by us. It was collected as the aim of this project was to generate popular music and related data was required. This was done by sourcing for the names of the top 100 video game soundtracks on YouTube. They were chosen based on the best reviews and highest number of likes. The selected songs were then downloaded using Musescore in the MIDI format, which was the same as the VGMIDI dataset.

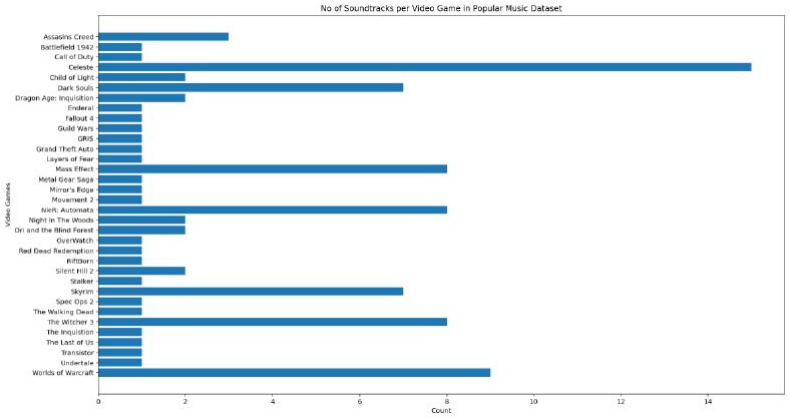

Figure 2: Bar chart of the number of soundtracks per video game in the popular music dataset

These resulted in a total of 302 songs in the MIDI format in the full combined dataset.

3 Data pre-processing

The individual pieces in our dataset were transposed into A, Bb, C, D, E, F, G major. This was done to introduce variations in the original dataset by increasing or decreasing the pitch of the songs as well as to multiply our dataset. This allowed the augmentation of our current dataset, increasing its size from 302 to 2114 songs.

The music pieces in the dataset were all in MIDI format, and Python's music21 toolkit developed by MIT was utilized to conduct further processing and analysis as it provided a simple interface to acquire the musical notation of MIDI.

The following are the different input features for the various models that were used to generate music.

3.1 Markov Chain

The Markov chain model served as an exploration and only one feature, the notes, was chosen as an input. The chords were broken down into individual notes. These notes were represented in the MIDI numbers from 0 to 127 and stored in an array following the music sequence. An order which considered the number of notes used to determine the next note was then decided and a transition matrix was built based on that order. To generate the music, an initial starting state with the length of the order was given and the transition matrix was used to predict the next note. If there were no past event that could be used to predict the next note, another starting state would be randomly generated.

3.2 Naïve LSTM and Locally Connected Convolutional CNN

Both the naïve LSTM and locally connected convolutional CNN models used two input features, notes and chords. These were obtained by parsing the midi files into music21's convertor which gave a stream object consisting of a list of all notes and chords. The notes remained as they were and the chords were encoded by the pitch classes of every note in the chord in a single string, with each pitch class separated by a dot. The octaves for the notes in the chord was not considered. This made our dictionary created later smaller. The notes and chords were appended to an array in order. These encodings allowed easy decoding of the output generated by the network into the correct notes and chords.

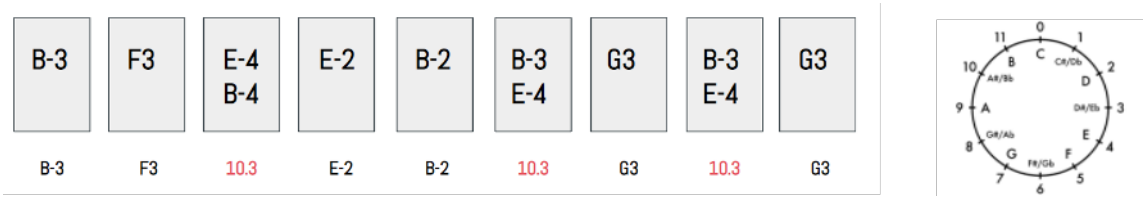

Figure 3: Representation of Notes and Chords

A dictionary was created to map the array entries to integers as neural networks perform better on numerical data as compared to strings. An input sequence and output sequence were also constructed where a sequence length for input was chosen and the output is the first note or chord that came after that sequence. Reshaping was then done to obtain a compatible format for the model layers.

3.3 LSTM with more features

This LSTM model had 3 input features, the individual notes, the first touch (start of a note), and the continuation (continuation of first touch). The notes were represented in MIDI numbers ranging from 0 to 127 and the chords were broken down into their individual notes as well.

To represent the first touch and the continuation of first touch, a constant value to both first touch and continuation were assigned. Doing this allowed the distinguishing between a long note and multiple short notes that are played after one another [1]. For instance, if first touch is assigned to 1 and continuation to 0.5, the sequence 1 - 0.5 - 0.5 represented one long note while the sequence 1 – 1 – 1 represented three short notes. Following that, the length of each MIDI file was standardized to 150-time units. Since most notes were in quarters, or in the multiples of 0.25, the assumption that the shortest note duration of 0.25 was made.

Using these features, a matrix was created to represent our MIDI datasets. The x-axis of the matrix represented time (duration and offset), while the y-axis represented the notes. This matrix will look like a spectrogram with the dimension (number of MIDI files, length of each MIDI file, total number of notes).

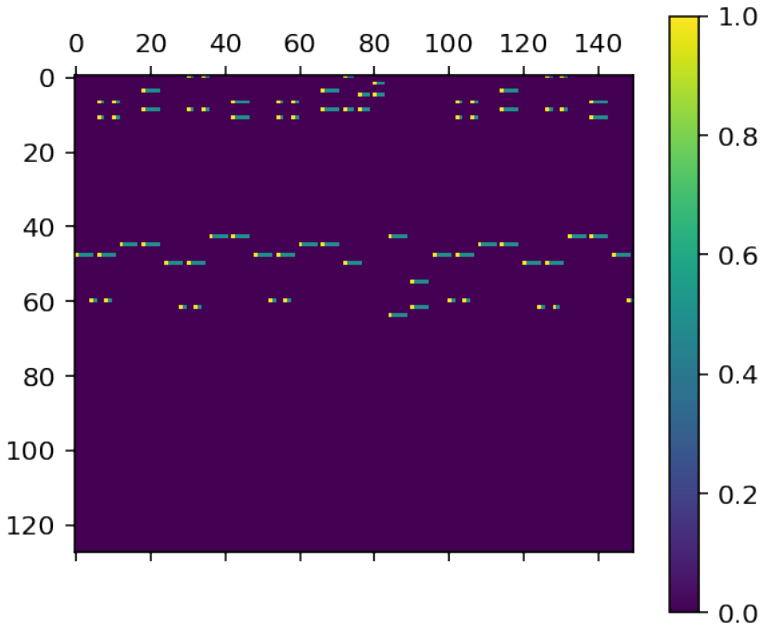

Figure 4: Matrix Representation of a MIDI File

To format the input for the LSTM model, the original matrix was first converted into (number of MIDI files*length of each MIDI file, total number of notes). Afterwards, the step size and the number of columns

that were taken into account to predict the next column were set. Two arrays were finally created for training, where the first array was used to predict the next array, with the second array being the true array.

4 Algorithm/Model

4.1 Markov Chain Model

Music is a temporal art where the order in which events happen is meaningful. The next note will depend on the previous note, it cannot be jumping five octaves higher. Conditional probability captures this concept, by quantifying the likelihood that an outcome will occur given that another outcome has already occurred. Thus, the Markov chain, a probability system, was used to model the sequence of musical events as it is able to express relationships between the future and past.

An order for deciding the number of past events was determined and the transition matrix containing the conditional probabilities was generated based on the order. Our goal was to use an order that is as small as possible, else the music generation would depend on a sequence of too many exact notes which might not exist in the transition matrix. This will result in many randomly reinitialized starting points and the music piece would not flow. However, the order could not be too small either as it would cause the prediction of the next note to be almost random.

A range of numbers for the order was tested to generate the music pieces. We observed that regardless of the numbers we chose, the model did not perform well, with chains of repeating notes even after experimenting multiple times.

The usage of Markov chains has also been used by others in the community to generate music, with results that were similar to ours. Lin (2016) noted that musical compositions via Markov chains did not sound good [4] and Zhang (2016) concluded that it was most probable for a note to come after itself [10]. Even after forcing the program to disregard the probability of a note repeating and use the next highest probability, it ended up with a pattern of two notes alternating. Higher order Markov chains also shared the same problem.

Hence, we decided to discontinue working on this model as it was too dependent on the exact states of past events which would not be good in the generation of new music. While the Markov chain is able to record the probabilities of note sequences, a better algorithm is still needed to generate a music piece with more realistic variation of notes.

4.2 Naïve LSTM and LSTM

Long Short-Term Memory is a type of Recurrent Neural Network that is capable of learning long-term dependencies and is well-suited for processing and making predictions based on time-series data. It solves the issue of a lack of long-term memory found in regular RNNs, which is accomplished by adding a specific memory cell. LSTM helps in recognizing melody structure and proper note sequences, which is crucial in generating meaningful music where the beginning of a song plays a role in the middle and the end.

The use of artificial neural networks like RNN and LSTM have become very popular for music generation. The main advantage of using artificial neural networks, LSTM in particular, is its capability to keep track of the musical notes used in the past, and utilize that to predict the new musical notes. Nayebi et. al (2017) developed an LSTM model which takes in a large corpus of text consisting of popular songs from David Bowie (a rock artist) and Madeon (an electronic dance music artist) for training [5]. For our project, we would like to utilize other methods of music representation, such as the music21 representation and using an array of matrices that represent the notes, first touch and continuation of the notes.

4.2.1 Naïve LSTM

This model was used as a starting point to help us gain more perspective and understanding of the expectations of using an LSTM model. The model architecture consists of one LSTM layer, followed by a dropout layer to prevent overfitting. The loss function was calculated using categorical cross entropy and the loss is minimized using Adam optimizer. The output is passed into a softmax layer for the final output.

This model did not perform well as the resulting music has a sequence of repeat notes. We suspected that the sequence of repeating notes might be due to the lack of representation of other musical features such as offset and duration. Hence, for the next LSTM model, we tried to incorporate these features into our input data.

4.2.2 LSTM with more features

For the LSTM model, the model architecture consisted of multiple LSTM layers to increase network capacity. Each of the LSTM layers is followed by a dropout layer that helps to regularize the training and prevents overfitting from occurring. This is done by randomly disconnecting a percentage of the synapses between the two layers. Softmax activation layer was added for each output.

The loss function was calculated using categorical cross entropy and the loss is minimized using rmsprop optimizer since that is what most RNN based models use. Similar to the previous model, Softmax activation layer was added for each output.

This model did not perform as well as we hoped as the resulting music of this model still has a sequence of repetitions, particularly for notes E5 and F5. Due to this, we decided to look into other feasible models that can be used to generate more meaningful music.

4.3 CNN

Convolutional neural networks have been recently used in music generation tasks. Some examples of existing work include MidiNet, which generates melody one bar after another [9], PerformanceNet, which generates music from music scores [8], and the WaveNet model proposed by DeepMind, which generates one audio sample at a time [6]. Of these, music generated by MidiNet performed comparably to Google's MelodyRNN model and was reported to be more interesting than the MelodyRNN music [9]. This showed that convolution architectures could possibly outperform recurrent neural networks (RNN) in tasks such as audio synthesis. This result is promising as CNNs are typically faster to train and are more easily parallelizable than RNNs. Thus, the CNN model could be suitable for our music generation task.

Sequences of length k are extracted from the training set as the input to this model. The model predicts the next note by using the previous k music notes in the sequence.

We first trained a naïve CNN model which consists of three 1D convolution layers, (one with 8 filters, one with 32 filters and one with 64 filters) with one stride, followed by a maximum pooling layer, a dense layer, and a softmax activation function.

The resulting music from the model followed some kind of pattern, which was good as music has repeating patterns. However, there were too many repeating notes, which results in the lack of chord progressions.

Since the naïve CNN model lacks the temporal feature, it was used as a baseline model and we tried incorporating Locally Connected CNN layers into our CNN architecture.

Locally Connected CNN layers allow the network to learn different type of features for different regions of the input. This allows us to learn different features at different parts of the music, where different phrases of in music may have different chord sequences.

Some variations of the CNN model with Locally Connected CNN layers were considered in this project.

4.3.1 Model A: 3 1D Locally Connected CNN layers

For Model A, all three 1D Convolution layers were replaced with Locally Connected Convolution layers, each with stride 1. 8 filters were used for the first layer, 32 filters for the second layer and 64 filters for the third layer.

4.3.2 Model B: 1D Locally Connected CNN layer followed by 2 1D Convolution layers

For Model B, the first convolution layer was replaced with a Locally Connected Convolution layer with 8 filters. This was followed by a 1D Convolution layer with 32 filters, and another 1D Convolution layer with 64 layers. All of these layers had stride 1.

4.3.3 Model C: 2 1D Convolution layers followed by 1D Locally Connected CNN layer

Model C consists of two 1D convolutional layers (one with 8 filters and one with 32 filters) with stride 1, followed by a 1D Locally Connected CNN layer which has 64 filters and stride 1.

5 Results

Model	***Loss***
Naïve LSTM (baseline)	1.7150
LSTM with more features	1.7918e-07
Naïve CNN (baseline)	3.1836
Locally Connected CNN (A)	1.1079
Locally Connected CNN (B)	3.1907
Locally Connected CNN (C)	0.0416

Figure 5: Comparison of training loss of all the models

Naïve LSTM model music produced by baseline performed better overall, as it had lesser repeating notes as compared to LSTM with more features, despite having a higher loss. An interesting observation from these two models is that in the Naïve LSTM model, there is a repeating sequence of patterns, whereas in the LSTM model with more features, there is repeating sequence of the same two notes.

For the CNN models, the results showed that using locally connected layers generally improved the performance of the baseline Naïve CNN model. This was because the layer allowed the network to learn different types of feature for different regions of the input which is better suited to music since it also contains different sequences such as major chords. That being said, model B's loss was similar to that of the Naïve CNN. This might be due to it having less weights as compared to A and C since its locally connected layer was in the first layer.

An interesting observation was that the music generated by models with lower loss had lesser repetitions of notes and more repeating patterns compared to those with higher losses. As such, those pieces sounded better and model C with the lowest loss performed the best during our evaluation. However, the disadvantage of using such a locally connected layer is that it could increase the number of parameters drastically and over fitting might occur if the dataset is limited.

6 Evaluation Methodology

Since our problem statement of generating the next popular music is subjective, there is no objective evaluation indicator to evaluate the music generated by each model. Hence, we first use the time series plots of our music pieces to evaluate if our AI generated music inherited the underlying patterns that characterize popular music pieces.

The time series plots allow the visualization of the note transition across time and the pitch range of the music. Following which, evaluated the AI generated music was evaluated with 3 criteria. First, the music piece should not have repetition of notes, as a music piece with notes repetition sounds relatively simple and boring. Second, the music piece should instead have repetition of sequences. This is required to generate the chorus of the music piece. Third, the music pieces should be contained within a pitch range. This is based on our insights upon analyzing our training dataset – the next note and current note played tends to be in the same pitch range. This is logical as the notes played by the pianist right and left hand are typically located within the same pitch range. Moreover, this enables a smooth transition between notes.

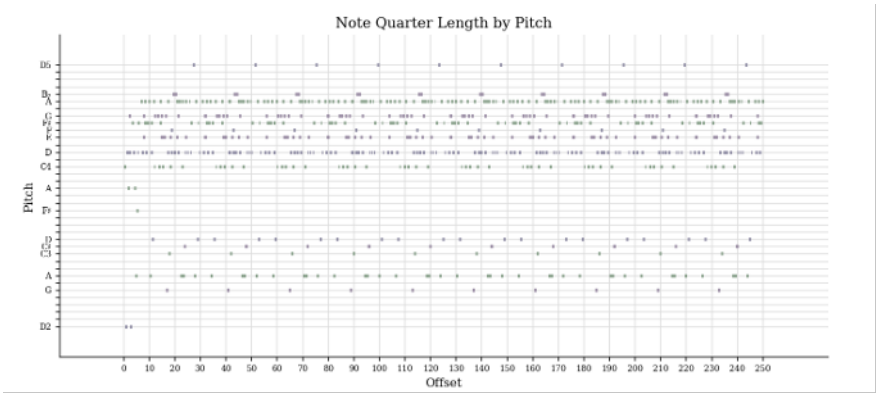

Figure 6: Naïve LSTM

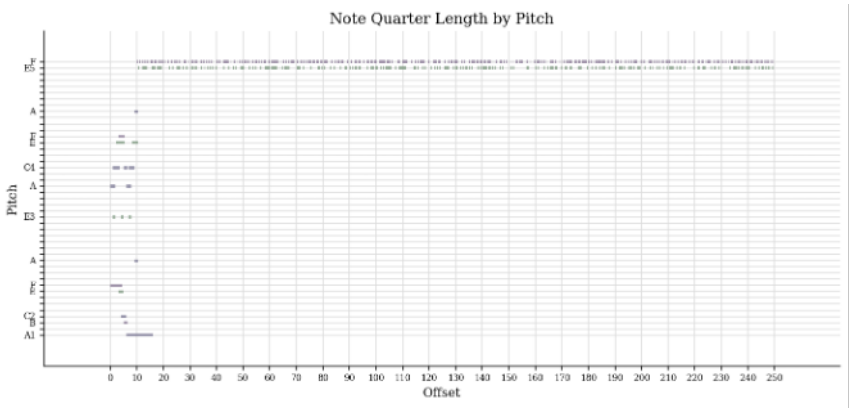

Figure 7: LSTM with more features

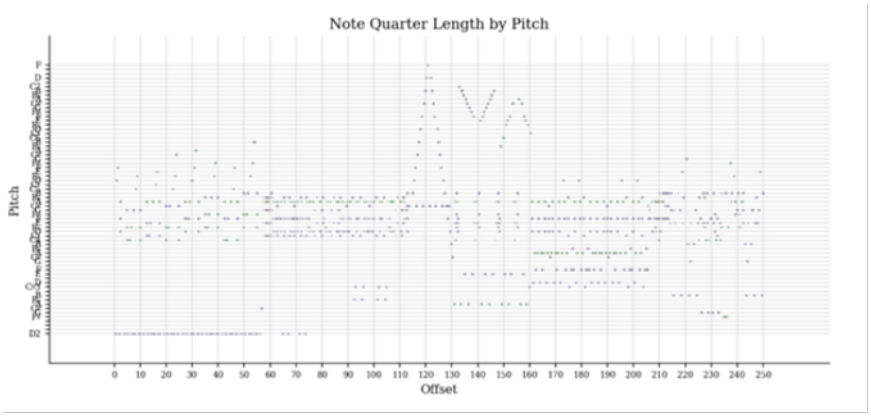

Figure 8: Locally Connected CNN (C)

Naïve LSTM:

There is a clear repetition of note sequences, which included chords. This gives authenticity to the music generated. Yet, there are no additional variances of notes or surprises between the repeated note sequences that causes the listeners to lose interest after some time.

LSTM with more features:

There is a slight variance in the notes played for the first few seconds of the music generated. However, the rest of the song consists of a repetition of two notes, E5 and F5. This causes listeners to lose interest very quickly and do not enjoy the song as much as the other songs.

Local Conv (C):

There are repeating sequences observed and minimal note repetition. The notes also transition smoothly and which gave the music piece an edge over the other two models. An interesting observation was that listeners liked the large variance in pitches.

Compared with using quantitative indicators for analysis, human listeners can evaluate the music pieces more holistically. In addition, it ties in with our problem statement, where we define popularity by ratings and reviews of the music pieces given by people. Hence, we conducted a survey, which aims to determine if our AI generated music can be the next popular music piece.

A total of 46 participants took part in the survey. They listened to 3 AI generated music, one from Naïve LSTM, LSTM with more features and Local Conv(C) each and 3 original popular video game songs from our collected dataset. Then the participants were required to rate each music piece with 1 being very bad and 5 being very good. (Question 1) Following which, they were tasked to rank the 6 music pieces based on their personal preference. (Question 2) Inspired by the Turing test, participants were tasked to identify which of the 6 AI music piece was AI generated. (Question 3)

(Question 1) The results showed that the AI generated music had an average rating of 3.44 which is higher than the benchmark average rating set by the original popular video game songs sampled of 3.

	AI Generated Music			Original popular video game songs		
Rating	Naive LSTM	LSTM with more features	Local Conv(C)	Original 1	Original 2	Original 3
Average	3.96	2.39	3.96	3.23	2.85	2.91
Mode	4	2	4	3	3	3
Overall average	3.00			3.44		
Overall mode	3			4		

Figure 9: Breakdown of average and modal ratings per music piece

(Question 2) The results illustrated a clear preference for music 4, followed by music 2. Specifically, the general consensus on the ranking of the 6 music pieces are as follows (from highest liking to least preferred):

1. Local Conv(C)
2. Naïve LSTM
3. Original 1
4. Original 3
5. Original 2
6. LSTM with more features.

(Question 3) The result highlighted that music 3 to 6 had almost equal chances of being identified as AI generated music. This emphasizes that music 4 is on equal par with the original popular video game songs. Moreover, less than half of the participants believes that music 2 is AI generated when in fact it is AI generated. In fact, only 39% believed that music 2 was AI generated. This is lower than the average of 46.5% of participants who believed the 3 original popular video game songs were AI generated.

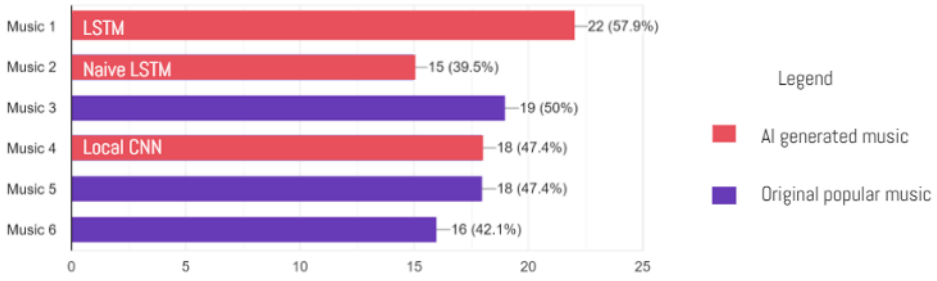

Figure 10: Percentage of participants who believe each music piece is AI generated

7 Discussion

7.1 Model Comparison

Music pieces generated by LSTM with more features have distinct note repetition problem. This explains the result observed in the survey where most participants were able to identify the music pieces as AI generated and is the least preferred music piece. Moving on, comparing the Naïve LSTM model with the Local Conv(C) model, both models had repeating note sequences and chords that made the music sound realistic. On top of this, Local Conv(C) has a segment of notes which illustrates a smooth transition of notes over a large pitch range. The ups and downs of music reflects the emotional change and brings excitement to the music.[3] As a result, music produced via Local Conv(C) is preferred over that of Naïve LSTM, as observed in the survey results. We then conclude that Local Conv(C) is our best model. The paper named Music Generation with Locally Connected Convolution Neural Networks had similar findings. They claimed Local Conv CNN model is observed to generate music with greater variance in notes [7].

7.2 Challenges faced

Two main problems were encountered. Firstly, note repetition was commonly observed in the music pieces generated. To resolve this, various attempts were made. They include shortening the music input sequence, including other features namely duration and offset and trying out new models. As illustrated above, locally connected convolution neural network worked best at encoding the music sequence and its stochastic nature allowed a more varied and lively music to be generated.

Secondly, finding the best music representation was difficult. Different music representations were fed into the Naïve LSTM model and the LSTM model. In the Naïve LSTM model, well established MIT music21 library was adopted to extract the pitches and chords. Whereas in the LSTM model, a matrix which included notes, first touch and continuation was used. Evidently, music21 music representation worked hand in hand with the LSTM model, generating more pleasant music pieces. A side note is the lack of explainability of the neural network models in general made it even harder to resolve the 2 main issues.

7.3 Future work

Despite the interesting results reached in our present work, many improvements could still be made. Currently, the models only produce video game piano music. We hope to be able to generate music with various styles and genre. One possible approach is to explore style transfer techniques which can turn a song from one style to another. To introduce creativity into the music generated, we would explore creative adversarial network.

We would also like to improve our model to be able to accurately represent all types of music. One of the limitations of our model lies in the assumption that was made when modelling the duration of the notes. This assumption ignores complex beats like triplets, which can add variation into music composition. Rest notes are also not included, which could more accurately represent common rhythms in other genres of music, such as swing and rock music.

Language and music have many similarities. Both consists of sequential events and follow a set of grammar rules. Similar to expecting the next word in a sentence, we can also expect the next note to be played in a song. Hence, we can explore adopting a similar idea of using word2vec to encode each slice of a music piece, where each slice is of the same duration and are non-overlapping. More importantly, Word2vec model can capture the tonal property of polyphonic music, which we believe will provide better features to our model.[2]

In addition, we could make our music generation process more adaptable and incremental, thereby mimicking the way a human composer mimics and generates music. Our neural network models are not adaptive as they only learn during training phase and are unable to adapt based on the feedback of users in the generative phase. For example, the negative feedback when the user stops listening to the music. Our neural network models are also not incremental as it generates the whole music sequence given an input sequence, which is different from the successive refinements that composer makes when producing music.

8 References

[1] Erdoan , H. (2018, April 27). Music Generation With Lstm. Hedonistrh. https://hedonistrh.github.io/2018-04-27-Music-Generation-with-LSTM/

[2] Herremans, D. (2019, May 7). Representing music with Word2vec? - Towards Data Science. Medium. https://towardsdatascience.com/representing-music-with-word2vec-c3c503176d52

[3] Huang, Y., Huang, X., & Cai, Q. (2018). Music Generation Based on Convolution-LSTM. Computer and Information Science, 11(3), 50. https://doi.org/10.5539/cis.v11n3p50

[4] Lin, A. (2018, June 1). Generating Music Using Markov Chains - HackerNoon.com. Medium. https://medium.com/hackernoon/generating-music-using-markov-chains-40c3f3f46405

[5] Nayebi, A., & Vitelli, M. (2015). Algorithmic Music Generation using Recurrent Neural Networks. Stanford University. https://cs224d.stanford.edu/reports/NayebiAran.pdf

[6] Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A Generative Model for Raw Audio. https://arxiv.org/pdf/1609.03499.pdf?utm_source=Sailthru&utm_medium=email&utm_campaign=Uncubed\%20Entry\%20\%2361\%20-\%20April\%203\%2C\%202019&utm_term=entry

[7] Ouyang, Z., Yin, Y., Yan, K., Wu, J., Hu, X., & Xin, S.-T. (2018). Music Generation with Locally Connected Convolutional Neural Network. http://ouyangzhihao.com/wp-content/uploads/2018/12/MUSIC-GENERATION-WITH-LOCAL-CONNECTED-CONVOLUTIONAL-NEURAL-NETWORK.pdf

[8] Wang, B., & Yang, Y.-H. (2019). PerformanceNet: Score-to-Audio Music Generation with Multi-Band Convolutional Residual Network. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 1174-1181. https://doi.org/10.1609/aaai.v33i01.33011174

[9] Yang, L. C., Chou, S. Y., & Yang, Y. H. (2017). MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. arXiv preprint arXiv:1703.10847.

[10] Zhang, J., & Boppana, P. (2016). Markov Chain Music Composition. http://www.math.utah.edu/~gustafso/s2017/2270/projects-2016/zhang-bopanna/zhangJie-bopannaPrathusha-MarkovChainMusicComposition.pdf