Carleton University

COMP 4107 - Final Project

Game Soundtracks Generation with Long Short-Term Memory Network

Tri Cao - ID: 100971065

Date

April 13, 2020

Contents

Contents . . . . . **i**

1 Abstract **1**

2 Introduction **1**

3 Background **1**
3.1 Understanding of music data 1
3.2 Recurrent Neural Network and Long Short-term Memory Units 2

4 Dataset **3**
4.1 Data Source 3
4.2 Preprocessing 3
4.3 Vectorize and embedding 3

5 Approach **4**
5.1 Network Architecture 4
5.2 Training method 4
5.3 Music generation method 4

6 Results **4**
6.1 Training metrics 4
6.2 Analyze generated tracks 5
6.2.1 The music score 5
6.2.2 Real audience scores on quality 5

7 Discussion **8**

8 Conclusions and Future Work **8**

9 Appendix **8**
9.1 Links to generated tracked by the model 8
9.2 Listening test responses 8

Bibliography **9**

1 Abstract

In the deep learning field, music generation problem has gained a lot of popularity in the recent years. Recurrent neural networks (RNN) have been proven to efficiently predict sequential music patterns, that the networks of this kind can store past memories to predict the future events. In this project, we use the long short-term memory (LSTM) [6], one of the most popular RNN architechtures, to train a collection of game soundtracks and perform music generation. We conclude that the network is capable to generate game-like tracks, which are fast-paced and highly repetitive. The music seems to sound better the longer the network is trained, and even capable of producing a long sequence from original soundtracks.

2 Introduction

The past decade has witnessed a lot of AI breakthroughs in all of aspects of life, and even in the artistic pursuits. In the music literature, deep network architechtures have been invented and able to compose indifferentiable tracks from the man-made ones. Google has developed Magenta [1], an open source research project specialized in generating music using various network infrastructures and training techniques, from LSTM to using reinforcement learning to generate more realistic tracks. Christine [2] has introduced MuseNet, a deep model network that can serve a two hour online concert for millions of audiences. These become our main motivations for the work presented in this report, which is to build a simple LSTM network that is capable of generating short game-like soundtracks. Our main focus is not to have our simple model generate music sequences that can really compete with the state of the art models, but to build a good first experience on an end-to-end process of an intelligent music generation task, as well as gaining practical knowledge on recurrent neural network, which we didn't have any previous work in the course COMP 4107. The main process of this work contains 3 big phases:

- **Searching and making training dataset**: We downloaded a collection of game music as midi format from [14], transformed them to a simplified version with contain only piano as the sole instrument, and vectorize the notes to get the train dataset
- **Build the model and training**: We use a two-layer LSTM [6] network with dropout to train the dataset and store the checkpoints for during training for evaluation.
- **Generate music and evaluate result**: Using a collection of seed notes to start with, we continuously use the model to generate various sequences of 500 notes and decode to midi file. We use the weights stored from different number of epochs while training to compose the songs and compare their performance.

The following sections will go through the background knowledge of music data and a brief description of LSTM network, we will into more details the 3 phases mentions above. Finally, we conclude the document with discussion about the results achieved, the takeaways and possible future work to extend this study.

3 Background

3.1 Understanding of music data

A songs is a complex sequential composition of a lots of music elements. Comparing to text, another type of sequential data, where each element in the sequence is just a word in the vocabulary, many elements can play at the same time for a music piece. We can have notes, chords, and instruments play together, and each element also has an on and off duration. This makes encoding music data exponentially more difficult. Thus in this work, we have chosen the music from one instrument (piano only), and simplified other elements to create trainable data. The following is the summary of important elements to understand project:

- **Note**: notes represent the pitch and duration of a sound in musical notation [13]
  - **Pitch**: the high or low of the sound of a note

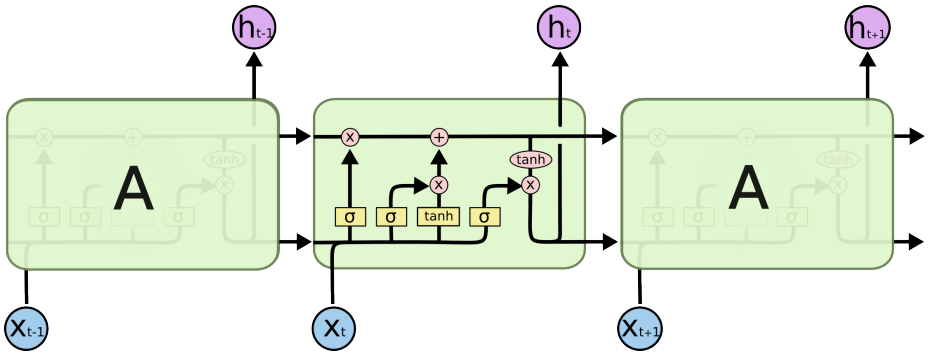

Figure 1: A typical LSTM structure (image obtain from [3])

  - **Duration**: the duration during which a note is played
- **Chord**: chords is a harmonic set of notes that are played at the same time [9]
- **Melody**: A linear sequence of musical tone that listener perceives as a single entity [10]
- **Monophony**: the simplest of musical textures, where a music piece only has one single instrument played, with one note at a time without accompanying harmony or chords [12]. In this work, our model is only capable of composing monophony tracks.

3.2 Recurrent Neural Network and Long Short-term Memory Units

In deep learning, recurrent neural network (RNN) was invented to solve the problem of learning sequential data, the next value in the sequence depends on the previous ones. A simple RNN or Elman network [5] contains an input layer, a hidden layer and an output layer for each time state, where the input $x^{(t)}$ depends on the hidden state value $s^{(t-1)}$ [8]. This simple RNN suffers from the vanishing gradient problem. The gradients, going through the long sequential hidden layers during back propagation, get smaller and smaller, making insignificant weight updates toward the beginning layers. Thus, the RNN seems to have shallow memory, where events deeper in the past, which may contain important information, fail to effect the prediction.

Long Short Term Memory Networks (LSTMs) were invented to solve this problem. Having similar deep sequential interacting layers like RNN, but the repeating modules have much more complex structure (figure 1). At each time step, in addition to the hidden state $h^{(t)}$, the network also stores a cell state $c^{(t)}$, which acts as a highway for the gradients to flow through the network, without the loss of gradient information. To maintain and control the information in the hidden states and cell states each memory module makes use of three memory gates. The first is the forget gate, which controls what information will be thrown away from the cell state. Secondly, the update gate is responsible to decide the memory to store in the cell state. Finally, the output gate decides what information from the cell state is used for the next prediction. LSTMs have been proven as an efficient solution for mitigating the gradient vanishing problem, and they are one of the most popular solutions to train sequential data.

$\tilde{c}^{(t-1)} = tanh(W_c[h^{(t-1)}, x^{(t)}] + b_c)$
$r_u = \sigma(W_u[h^{(t-1)}, x^{(t)}] + b_u)$
$r_f = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f)$
$r_o = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$
$r_o = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$
$c^{(t)} = r_u\tilde{c}^{(t)} + r_f c^{(t-1)}$

$\tilde{c}^{(t)} \rightarrow$ tanh component for cell state
$c^{(t)} \rightarrow$ Cell state
$r_u \rightarrow$ Update gate
$r_f \rightarrow$ Forget gate
$r_o \rightarrow$ Output gate

$h^{(t)} \rightarrow$ Hidden state at time t
$x^{(t)} \rightarrow$ Input at time t

4 Dataset

4.1 Data Source

We download 728 game soundtracks from [14]. All of the tracks are in the midi format [11], an old school technical standard for encoding wide variety of electrical musical instruments. However, all the tracks only contain piano as the sole instrument, thus simplify the training task.

4.2 Preprocessing

Processing the midi files is a challenging task. Even if the tracks are only performed by piano, but for each time step there are too many variables to keep track of, including multiple notes and chords playing at the same time, each may have different durations. The combination of all these variables can lead to the embedded vectors with huge number of dimensions, making the learning of the model unrealistic with our computing resources. Therefore, we decided to sacrifice some variations which are not directly related to the melody, with the trade off for the generation quality later on, to create a embedded training vectors:

- Transform the tracks to single instrument (piano)
- Only keep the main note from the chords
- If there are multiple notes being played at the same time, only keep the highest note
- Remove all the overlapping notes, which means the next note will start only when the current note has finished playing

As the results, we have transformed all tracks in to a low-quality monophony version, while keeping the melody untouched as our best. In the context of python coding, we use the music21 [4] to process the midi files

4.3 Vectorize and embedding

We use melody-rnn borrowed from magenta [7], a simple method to represent a MIDI note sequence to vectorize into a vector of integers. The notes in the sequence are represented as a sequence of integers (figure 2):

- 0-127: play a note with the respective MIDI note (60 for C in the octave 4)
- 128: stop the currently playing note (thus represent whole duration of a note)
- 129: do nothing (keep silent or keep the playing note on)

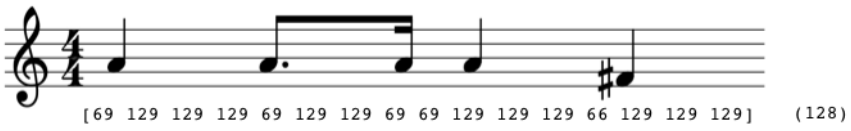

Figure 2: Encode a music score into a sequence of integer using melody-rnn [7]

With this simple rule, each song will be transformed into a sequence of integers, each integer in the range of 0-129. Each number in the sequence can then be feed to a embedding layer, which transforms a number to a one hot vector of 130 dimensions. We then decide that each note will be generated by a sequence of 30 previous note. For each song we break the notes downs to 31-grams sequences, which 30 first notes to create the training sequence and the 31st note being the label to be predicted by the model. Running the slicing window across all the songs, obtain a complete ready-to-train dataset.

5 Approach

5.1 Network Architecture

We use a 2-layer LSTM network to create train the model. After each LSTM layer, we apply a dropout layer to combat overfitting. Finally, we feed the output from LSTM layer to another 2 fully-connected layers, then apply softmax to get the output probability distribution (figure 3).

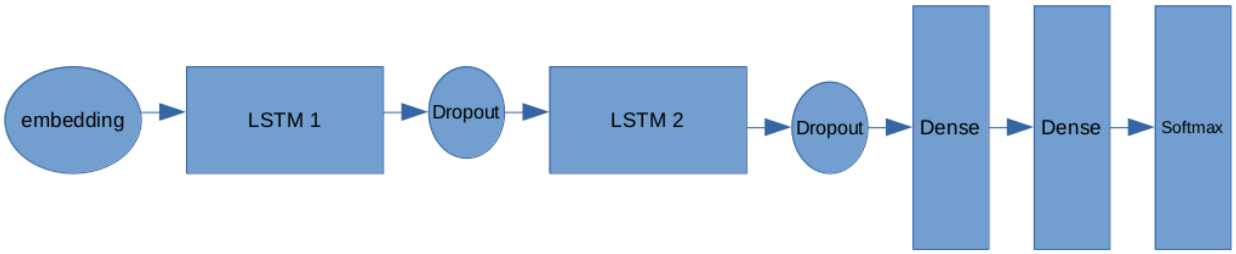

Figure 3: A graph of network architechture

5.2 Training method

We directly feed the training dataset obtained as mentioned in the previous section into the deep network, and apply the training for different epochs, using the Adam optimizer. To evaluate the performance of the network regarding the training time, we store the weights obtained from different training iterations to generate the music later on.

5.3 Music generation method

After training, the model is ready to generate music. We take a random 30-dimensional vector from the training dataset as the starting point, and let the model generate note by note until we have a sufficient number of notes for the song (500 notes for our cases). To allow variations in our songs, we don't just choose the note with the highest probability returned from the output vector. Instead we use the vector as a probability distribution to randomly generate the note, which helps us avoid too many repetitive patterns and boring songs. The generated sequence in the melody-rnn form will be decoded back using music21 and saved at midi format.

6 Results

6.1 Training metrics

We plot the accuracies and losses over 20 epochs of training (figure 4, 5)

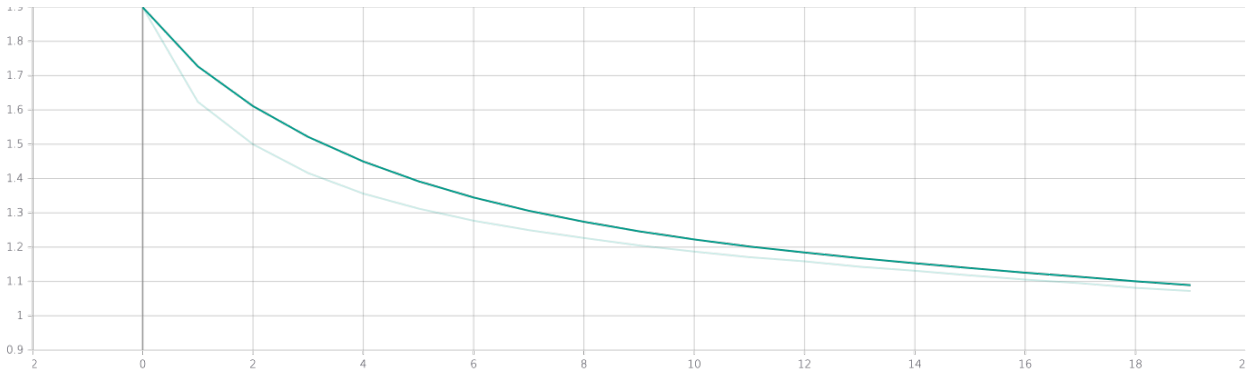

Figure 4: Losses over training epochs

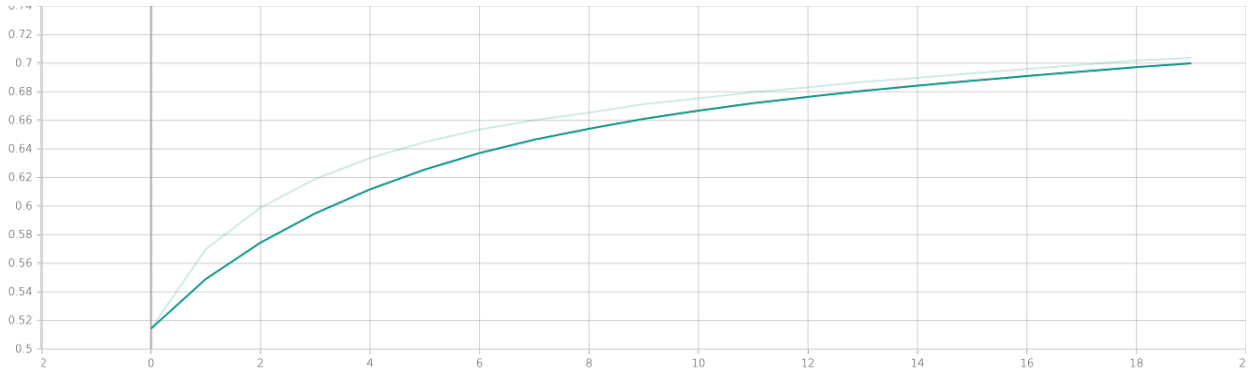

Figure 5: Accuracies over training epochs

6.2 Analyze generated tracks

To evaluate the performance of the model given different training epochs, we load the stored weights and let model generate the tracks.

6.2.1 The music score

Figures 6-9 show the sample music scores from the tracks generated by our model. With no training (figure 6), the output notes seems to be out of scale and totally random. This randomness slowly decrease over longer training periods (figure 7 and 8). At 20 epochs, we can see that the model has learned to generate long repetitive melody (figure 9). We even trained the network for even longer, and notice that the model can generate melody sequences directly from the original soundtracks. While this is a proof that the model was able to learn sequential data, it would be better to avoid the model directly copying from the original melodies.

6.2.2 Real audience scores on quality

We also created a survey and received the feedback from listening evaluations on the generated tracks on 17 audiences (see appendix). There were 4 songs listened and voted, with the varying training epochs (0, 20, 40 and 60 epochs). For each song, a score from 1 to 5 is given. The higher the score, the better the song according to the listener.

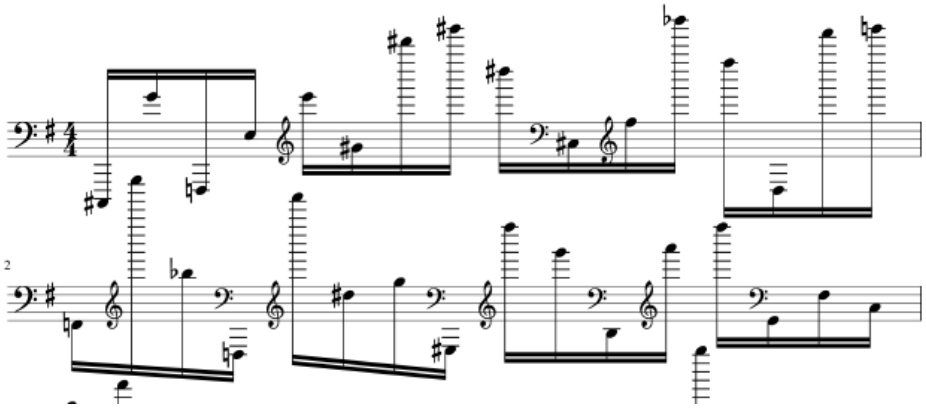

Figure 6: Music generated with no training

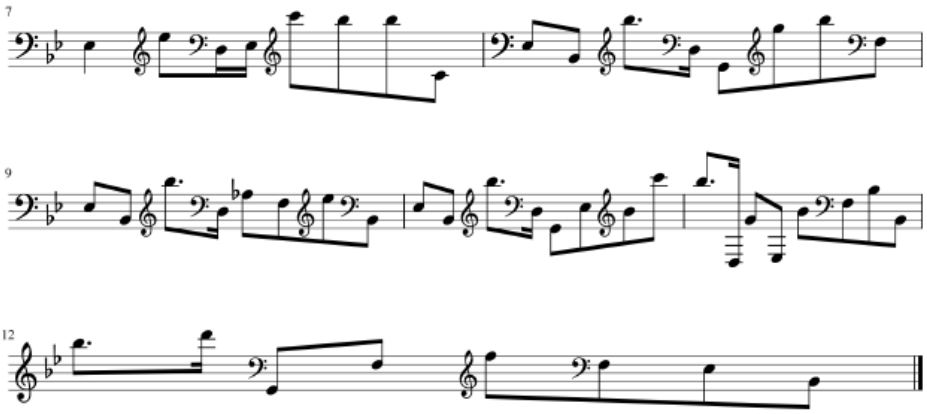

Figure 7: Music generated after training in 5 epochs

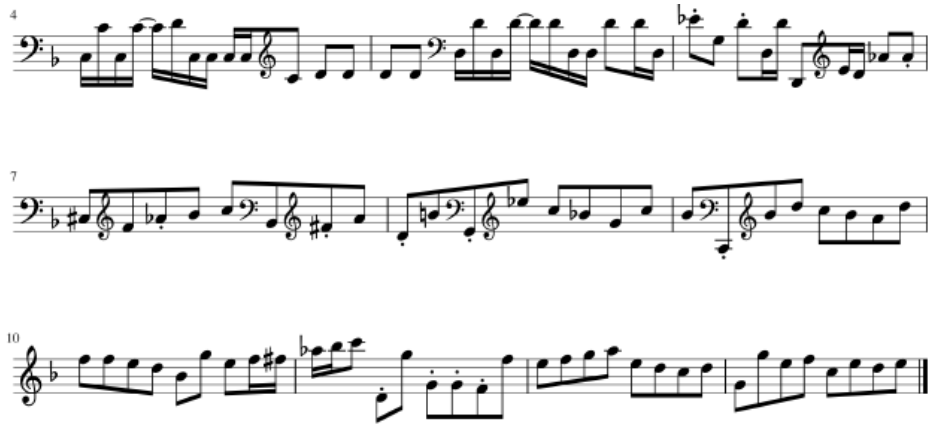

Figure 8: Music generated after training in 10 epochs

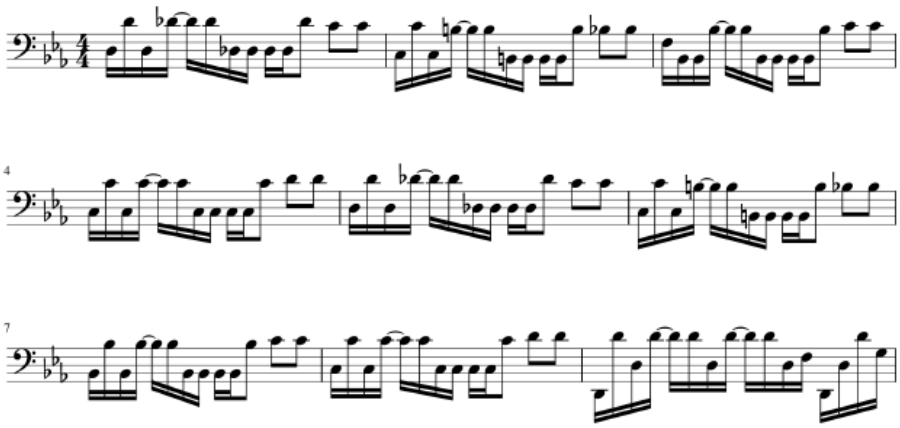

Figure 9: Music generated after training in 20 epochs

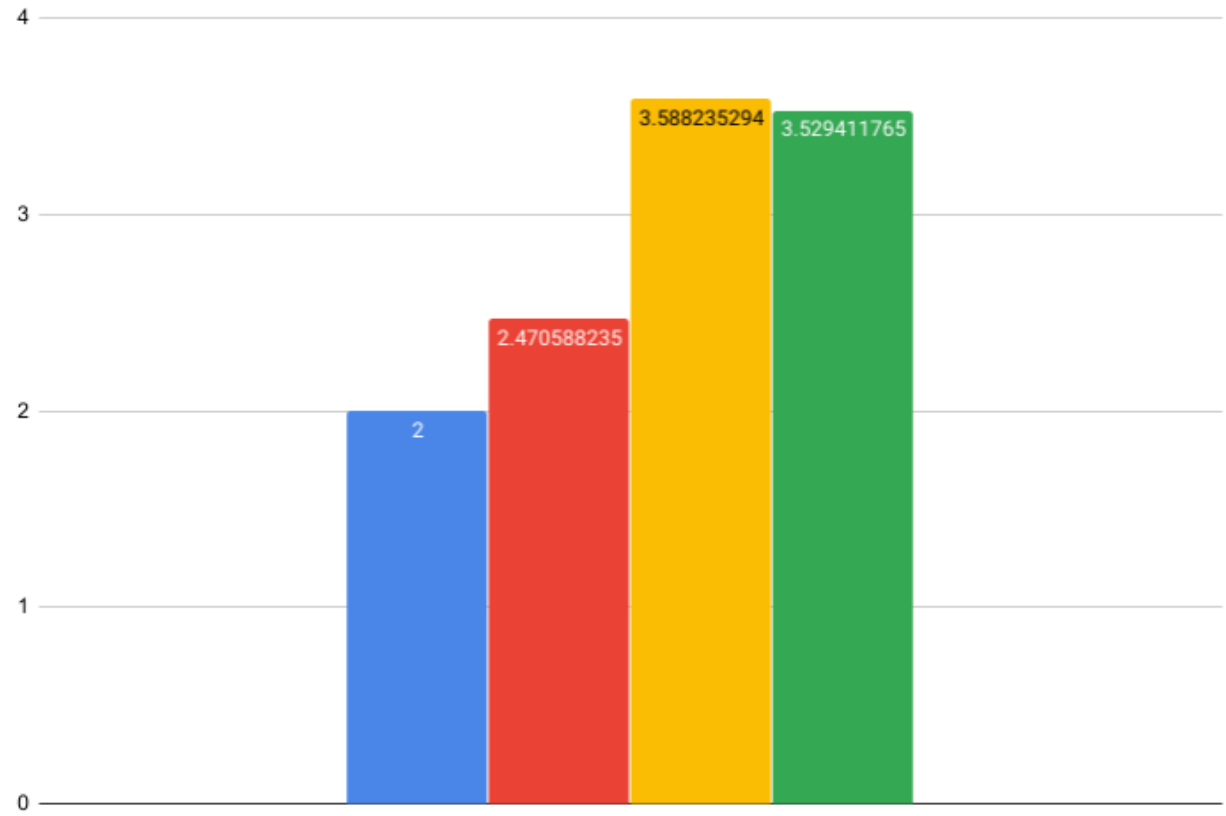

Figure 10: From left to right: average user scores from song 1 to song 4. Song 1 was generate at 0 epoch, song 2 at 20, song 3 at 40 and song 4 at 60 epochs

7 Discussion

Generally, we have successfully obtained the goal of testing the performance of a simple LSTM network in this project. The model seems to get better and better at generating music over longer period of of training, and it is able to remember long sequences from the original training songs. From the totally random output, the LSTM network can generate melodies that have a lot of repetitive patterns like the training game soundtracks. In addition, from the evaluations obtained from the survey and from our own, the generated results seem to resemble the game soundtracks style pretty well. For our knowledge, we have verified that LSTM network is a very efficient model way to predict sequential data, and successfully mitigates the vanishing gradients problem.

In the other hand, the obtained results are not any close to real compositions from human. Due to our limitation in time and resources, we have sacrificed some of the valuable information on the original music data, thus our generated tracks are just very simple monophony songs. Moreover, we didn't use any method to control the output from the model, thus the tracks do not have a smooth beginning and ending. Finally, training the model for big number of epochs also make the model just copy some exact outstanding melodies from the original tracks. Although this shows that the model was able to learn, copying is not a favorable behavior for composing music or other arts.

8 Conclusions and Future Work

We have gained valuable experience building deep recurrent model to generate music. First, we downloaded, simplified MIDI game soundtracks, then vectorized the tracks into trainable data for our model. Next, a two-layer LSTM model was trained for different number of epochs. After training, the model was able to generate game-like melodies using a random starting seed sequence.

There are lots of rooms to improve the music generation task. First, we could choose a different encoding method that adds the information about chords and other instruments into the model, so that the generated songs have richer musical textures. Secondly, we can experiment with more advanced generative networks, which are more suitable for the music domain. Finally, we can apply more controls and restrictions to the output notes using reinforcement learning, so that generated melodies are more natural and closer to human-made music.

9 Appendix

9.1 Links to generated tracked by the model

Sound Cloud link to the playlist: https://soundcloud.com/cao-minh-tri-149013147/sets/generated-by-ai

We also include the best songs as MIDI files in the submission

9.2 Listening test responses

Click the link below to see the result from the listening survey

Survey result: https://docs.google.com/forms/d/14O5f6y9Zd4Pv9MPYV5Iu2-usFKm0z11_pe0dItLdhaI/edit?usp=sharing

Bibliography

[1] Google Brain. Magenta project. URL https://research.google/teams/brain/magenta/.

[2] Payne Christine. Musenet., 2019. URL openai.com/blog/musenet. Online; accessed 11-April-2020.

[3] Christopher Olah. Understanding lstm networks. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Online; accessed 12-April-2020.

[4] Michael Scott Cuthbert. music21: a toolkit for computer-aided musicology. URL https://web.mit.edu/music21/. Online; accessed 12-April-2020.

[5] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Charles Martin. Music generation with an rnn. URL https://creativeprediction.xyz/hack/melody/. Online; accessed 13-April-2020.

[8] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010. URL http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10.

[9] Wikipedia contributors. Chord (music), . URL https://en.wikipedia.org/wiki/Chord_(music). Online; accessed 12-April-2020.

[10] Wikipedia contributors. Melody, . URL https://en.wikipedia.org/wiki/Melody. Online; accessed 12-April-2020.

[11] Wikipedia contributors. Midi, . URL https://en.wikipedia.org/wiki/MIDI. Online; accessed 12-April-2020.

[12] Wikipedia contributors. Monophony, . URL https://en.wikipedia.org/wiki/Monophony. Online; accessed 12-April-2020.

[13] Wikipedia contributors. Musical note, . URL https://en.wikipedia.org/wiki/Musical_note. Online; accessed 12-April-2020.

[14] Mike Newman (Yaginuma). Vgmusic. URL https://www.vgmusic.com/. Online; accessed 11-April-2020.