A Technical Report

On

Music Generation using Deep Learning

Submitted in partial fulfillment of the requirements of

Jawaharlal Nehru Technological University

In

COMPUTER SCIENCE AND ENGINEERING

By

Mohana Naga Sai Vyshnavi Medisetty

(17wh1a0567)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

Rajeev Gandhi Nagar, Nizampet Road, Bachupally HYDERABAD - 500 090

2017-2021

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Rajeev Gandhi Nagar, Nizampet Road, Bachupally HYDERABAD - 500 090

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the Technical Report titled **"Music Generation using Deep Learning"** is being submitted by **Mohana Naga Sai Vyshnavi Medisetty (17WH1A0567)** of the **IV B. Tech Computer Science and Engineering, (Academic year 2017- 2021)** during the **I semester** in partial fulfillment for the award of B.Tech in CSE to Jawaharlal Nehru Technological University, Hyderabad.

Seminar Coordinator
Ms. C Nagaraju
Assistant Professor
Department of Computer Science and Engineering

Dr. Srinivas Reddy Konda,
Head of the Department
Department of Computer Science and Engineering

ACKNOWLEDGMENT

Firstly, I would like to express my immense gratitude towards **BVRIT HYDERABAD College of Engineering for Women**, which created a great platform to attain profound technical skills in the field of Computer Science, thereby fulfilling my most cherished goal.

I would like to extend my sincere thanks and gratitude to **Dr. K. V. N. Sunitha,** Principal, BVRIT HYDERABAD College of Engineering for Women for her constant encouragement in carrying my Technical Seminar successfully.

My sincere thanks and gratitude to **Dr. Srinivas Reddy Konda,** Head of the Department, Computer Science & Engineering, for his inspiring guidance in carrying out the technical seminar presentation.

It is a privilege for me to record here my deep gratitude to **Mr. C .Naga Raju,** Assistant Professor**,** Computer Science & Engineering for helping me in collecting data of my chosen topic for the technical seminar; I would like to thank his co-operation and guidance throughout the preparation of the technical report.

Mohana Naga Sai Vyshnavi Medisetty

(17WH1A0567)

ABSTRACT

Composing music is a very interesting challenge that tests the composer's creative capacity, whether it is a human or a computer. Although there have been many arguments on the matter, almost all of music is some regurgitation or alteration of a sonic idea created before. Thus, with enough data and the correct algorithm, deep learning should be able to make music that would sound human. This report outlines various approaches to music composition through the Neural Network model, it is evident that musical ideas can be gleaned from these algorithms in hopes of making a new piece of music. The use of deep learning to solve problems in literary arts has been a recent trend that has gained a lot of attention and automated generation of music has been an active area. This project deals with the generation of music using some form of music notation relying on various LSTM(Long Short Term Memory) architectures. Fully connected and convolutional layers are used along with LSTM to capture rich features in the frequency domain and increase the quality of music generated. The work is focused on unconstrained music generation and uses no information about musical structure such as notes or chords to aid learning.

INDEX

S.no	Contents	Page no.
1	Introduction	1
2	Making sense out of sound	2
3	Music representations	3
4	Recurrent Neural Networks(RNN)	5
5	Vanishing and Exploding Gradient Problems	6
6	LSTMs	7
7	Music21	8`
8	Keras	8
9	Training	10
10	Preparing the data	12
11	Model	14
12	Generating music	15
13	Conclusions and Future work	16
14	References	17

LIST OF FIGURES

S.no	Contents	Page no.
1	Making sense out of sound	1
2	Sheet music	2
3	ABC Notation	3
4	MIDI Format	4
5	Recurrent Neural Networks	5
6	LSTMs	8

INTRODUCTION

This report focuses on generating music automatically using Recurrent Neural Network (RNN). We do not necessarily have to be a music expert in order to generate music. Even a non-expert can generate decent quality music using RNN.

We all like to listen to interesting music and if there is some way to generate music automatically, particularly decent quality music then it's a big leap in the world of music industry. Task: Our task here is to take some existing music data then train a model using this existing data. The model has to learn the patterns in music that we humans enjoy. Once it learns this, the model should be able to generate new music for us. It cannot simply copy-paste from the training data.It has to understand the patterns of music to generate music.We here are not expecting our model to generate new music which is of professional quality, but we want it to generate a decent quality music which should be melodious and good to hear.

Making sense out of sound

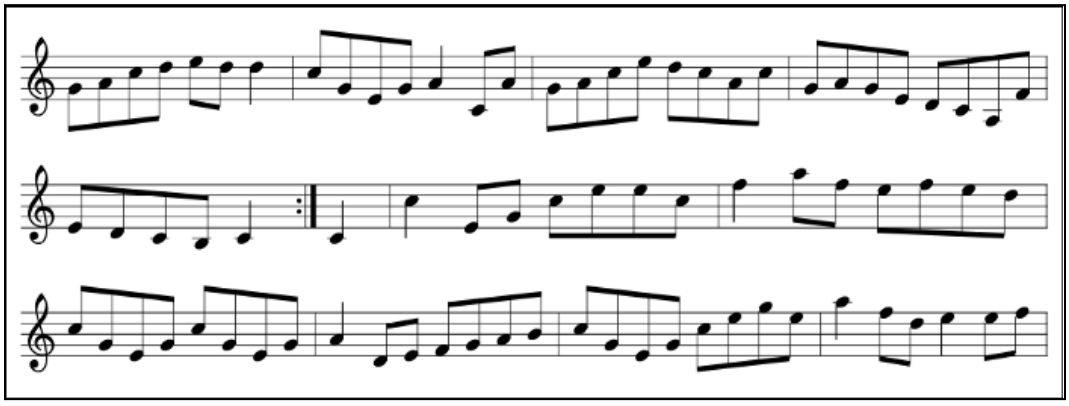
Music is a collection of tones of different frequencies. So, the Automatic Music Generation is a process of composing a short piece of music with minimum human intervention. One is able to compose his/her own original music without really knowing any music theory! We do not necessarily have to be a music expert in order to generate music. Even a non-expert can generate decent quality music using Deep learning models.

Constituent Elements of Music

- **Note**: The sound produced by a single key is called a note
- **Chords**: The sound produced by 2 or more keys simultaneously is called a chord. Generally, most chords contain at least 3 key sounds
- **Octave**: A repeated pattern is called an octave. Each octave contains 7 white and 5 black keys

Music Representations

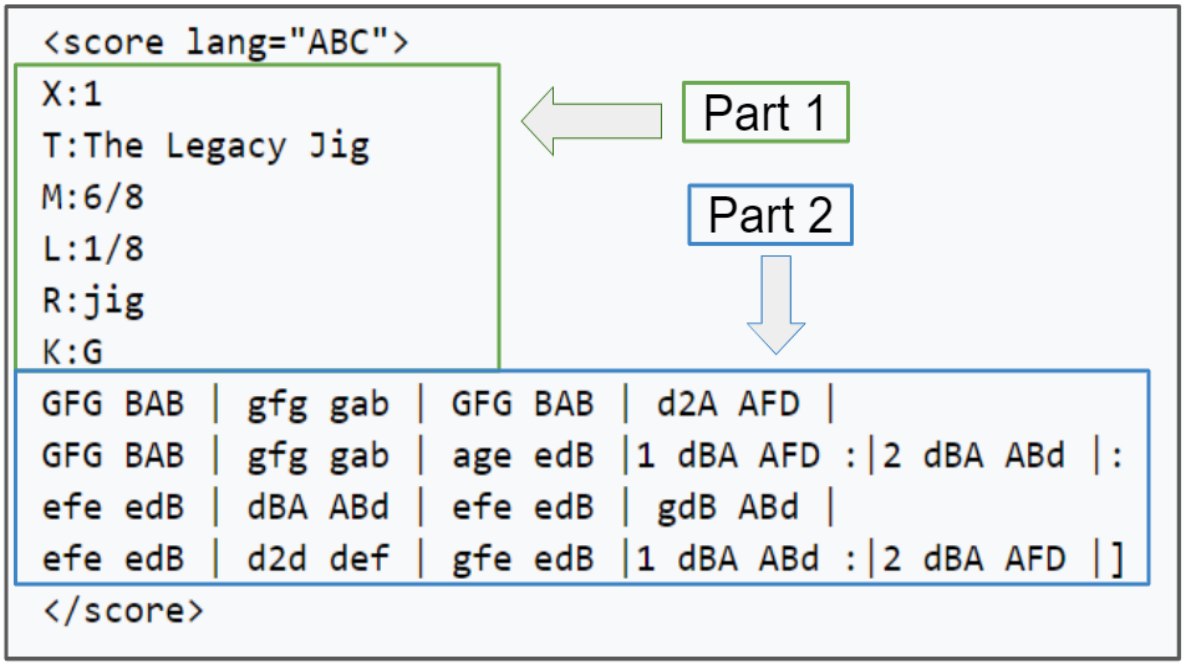
1. ***Sheet music*** **-** Music is represented by a sequence of musical notes. Each musical note is separated by a space. This can be used to represent both single instrument and multi instrument music.



2. ***ABC Notation*** - ABC notation is a shorthand form of musical notation that uses the letters A through G to represent musical notes, and other elements to place added values. These added values include sharps, flats, the length of a note, the key, and ornamentation.

   This form of notation began as an ASCII character set code to facilitate music sharing online, adding a new and simple language for software developers that's easy to use.

   Lines in part 1 of the music notation show a letter followed by a colon. These indicate various aspects of the tune such as the index, when there is more than one tune in a file (X:), the title (T:), the time signature (M:), the default note length (L:), the type of tune (R:) and the key (K:). The lines following the key designation represent the tune itself.



3. MIDI Format

The data splits into two object types: Notes and Chords. Note objects contain information about the **pitch**, **octave**, and **offset** of the Note. And Chord objects are essentially a container for a set of notes that are played at the same time.

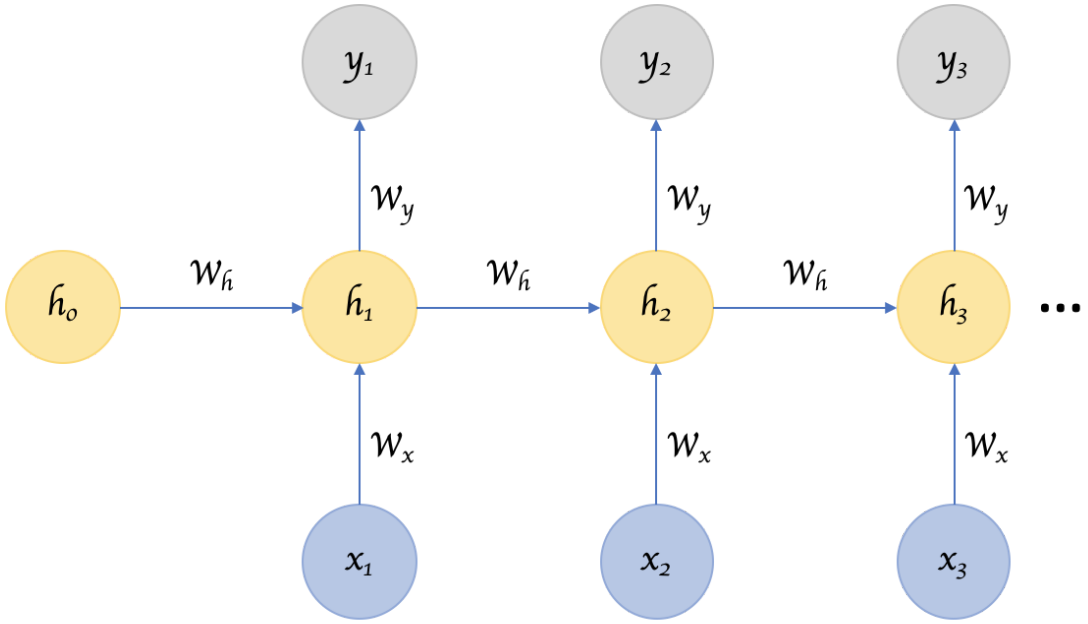
```
...
<music21.note.Note F>
<music21.chord.Chord A2 E3>
<music21.chord.Chord A2 E3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.note.Note G>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord A2 E3>
...
```

Recurrent Neural Networks

As music is a sequence of characters therefore the obvious choice will be RNN or variations of RNN like LSTMs or GRUs which can process sequence information very well by understanding the patterns in the input.

A recurrent neural network is a specific type of neural network which is based on sequential information and data. Reason behind calling them recurrent is because the same set of weights are applied recursively over a differential graph-like structure. RNN has a great application in Natural Language Processing.

RNNs add an interesting twist to basic neural networks. Unlike CNNs, RNN have a memory which remembers all information about what has been calculated. The main disadvantage of RNN is that they suffer from vanishing and exploding problems.



Vanishing and Exploding Gradient Problems

In a network of *n* hidden layers, *n* derivatives will be multiplied together. If the derivatives are large then the gradient will increase exponentially as we propagate down the model until they eventually explode, and this is what we call the problem of ***exploding gradients***. Alternatively, if the derivatives are small then the gradient will decrease exponentially as we propagate through the model until it eventually vanishes, and this is the ***vanishing gradient*** problem.

In the case of exploding gradients, the accumulation of large derivatives results in the model being very unstable and incapable of effective learning, The large changes in the models weights creates a very unstable network, which at extreme values the weights become so large that is causes overflow resulting in NaN weight values of which can no longer be updated. On the other hand, the accumulation of small gradients results in a model that is incapable of learning meaningful insights since the weights and biases of the initial layers, which tends to learn the core features from the input data (X), will not be updated effectively. In the worst case scenario the gradient will be 0 which in turn will stop the network and will stop further training.

LSTM Networks

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in the following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural networks. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

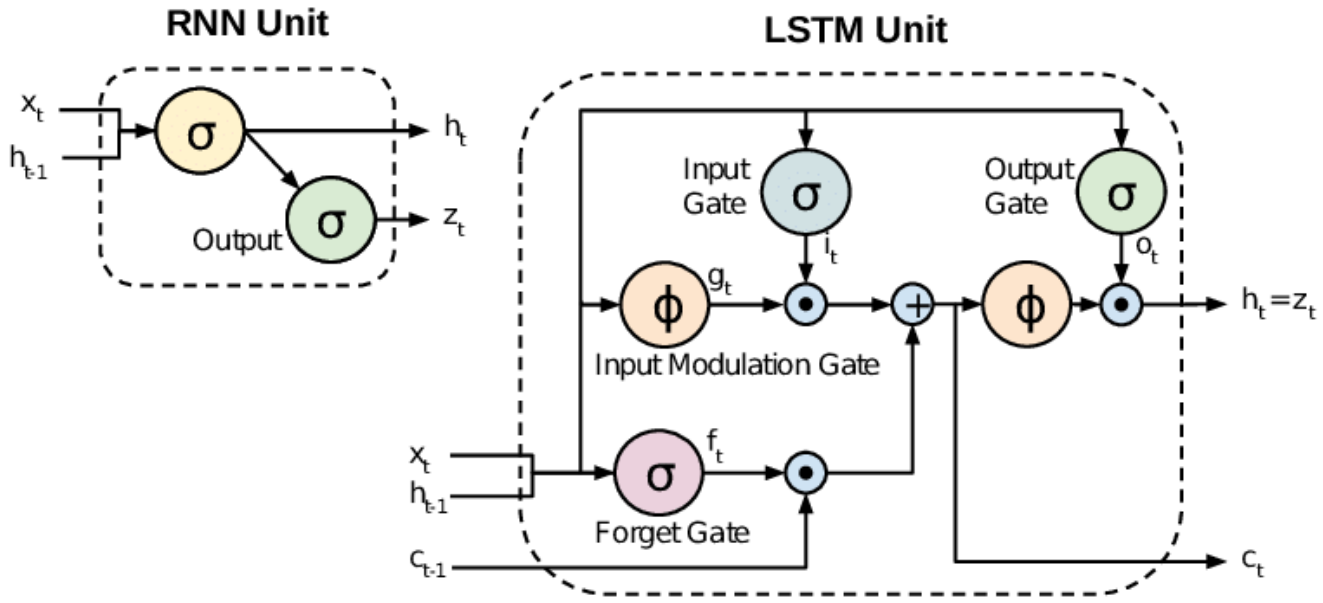
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.



Standard RNNs (Recurrent Neural Networks) suffer from vanishing and exploding gradient problems. LSTMs (Long Short Term Memory) deal with these problems by introducing new gates, such as input and forget gates, which allow for a better control over the gradient flow and enable better preservation of “long-range dependencies”. The long range dependency in RNN is resolved by increasing the number of repeating layers in LSTM.

Music21

Music21 is a Python toolkit used for computer-aided musicology. It allows us to teach the fundamentals of music theory, generate music examples and study music. The toolkit provides a simple interface to acquire the musical notation of MIDI files. Additionally, it allows us to create Note and Chord objects so that we can make our own MIDI files easily.

In this tutorial we will use Music21 to extract the contents of our dataset and to take the output of the neural network and translate it to musical notation.

Keras

Keras is a high-level neural networks API that simplifies interactions with Tensorflow. It was developed with a focus on enabling fast experimentation.

In this tutorial we will use the Keras library to create and train the LSTM model. Once the model is trained we will use it to generate the musical notation for our music.

Training

In this section we will cover how we gathered data for our model, how we prepared it so that it could be used in a LSTM model and the architecture of our model.

Data

In our Github repository we used piano music, mostly consisting of music from Final Fantasy soundtracks. We picked Final Fantasy music due to the very distinct and beautiful melodies that the majority of the pieces have and the sheer amount of pieces that exist. But any set of MIDI files consisting of a single instrument would work for our purposes.

The first step to implementing the neural network is to examine the data we will be working with.

The data splits into two object types: Notes and Chords. Note objects contain information about the pitch, octave, and offset of the Note.

- Pitch refers to the frequency of the sound, or how high or low it is and is represented with the letters [A, B, C, D, E, F, G], with A being the highest and G being the lowest.
- Octave refers to which set of pitches you use on a piano.
- Offset refers to where the note is located in the piece.

And Chord objects are essentially a container for a set of notes that are played at the same time.

Now we can see that to generate music accurately our neural network will have to be able to predict which note or chord is next. That means that our prediction array will have to contain every note and chord object that we encounter in our training set. In the training set on the Github page the total number of different notes and chords was 352. That seems like a lot of possible output predictions for the network to handle, but a LSTM network can easily handle it.

Next we have to worry about where we want to put the notes. As most people that have listened to music have noticed, notes usually have varying intervals between them. You can have many notes in quick succession and then followed by a rest period where no note is played for a short while.

Below we have another excerpt from a midi file that has been read using Music21, only this time we have added the offset of the object behind it. This allows us to see the interval between each note and chord.

```
...
<music21.note.Note B> 72.0
<music21.chord.Chord E3 A3> 72.0
<music21.note.Note A> 72.5
<music21.chord.Chord E3 A3> 72.5
<music21.note.Note E> 73.0
<music21.chord.Chord E3 A3> 73.0
<music21.chord.Chord E3 A3> 73.5
<music21.note.Note E-> 74.0
<music21.chord.Chord F3 A3> 74.0
<music21.chord.Chord F3 A3> 74.5
<music21.chord.Chord F3 A3> 75.0
<music21.chord.Chord F3 A3> 75.5
<music21.chord.Chord E3 A3> 76.0
<music21.chord.Chord E3 A3> 76.5
<music21.chord.Chord E3 A3> 77.0
<music21.chord.Chord E3 A3> 77.5
<music21.chord.Chord F3 A3> 78.0
<music21.chord.Chord F3 A3> 78.5
<music21.chord.Chord F3 A3> 79.0
...
```

As can be seen from this excerpt and most of the dataset, the most common interval between notes in the midi files is 0.5. Therefore, we can simplify the data and model by disregarding the varying offsets in the list of possible outputs. It will not affect the melodies of the music generated by the network too severely. So we will ignore the offset in this tutorial and keep our list of possible outputs at 352.

Preparing the Data

Now that we have examined the data and determined that the features that we want to use are the notes and chords as the input and output of our LSTM network it is time to prepare the data for the network.

First, we will load the data into an array as can be seen in the code snippet below:

```
from music21 import converter, instrument, note, chord

notes = []

for file in glob.glob("midi_songs/*.mid"):
    midi = converter.parse(file)
    notes_to_parse = None

    parts = instrument.partitionByInstrument(midi)

    if parts: # file has instrument parts
        notes_to_parse = parts.parts[0].recurse()
    else: # file has notes in a flat structure
        notes_to_parse = midi.flat.notes

    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in
element.normalOrder))
```

We start by loading each file

into a Music21 stream object using the *converter.parse(file)* function. Using that stream object we get a list of all the notes and chords in the file. We append the pitch of every note object using its string notation since the most significant parts of the note can be recreated using the string notation of the pitch. And we append every chord by encoding the id of every note in the chord together into a single string, with each note being separated by a dot. These encodings allow us to easily decode the output generated by the network into the correct notes and chords.

Now that we have put all the notes and chords into a sequential list we can create the sequences that will serve as the input of our network.

First, we will create a mapping function to map from string-based categorical data to integer-based numerical data. This is done because neural networks perform much better with integer-based numerical data than string-based categorical data. An example of a categorical to numerical transformation.

Next, we have to create input sequences for the network and their respective outputs. The output for each input sequence will be the first note or chord that comes after the sequence of notes in the input sequence in our list of notes.

Model

For each LSTM, Dense, and Activation layer the first parameter is how many nodes the layer should have. For the Dropout layer the first parameter is the fraction of input units that should be dropped during training.

For the first layer we have to provide a unique parameter called *input_shape.* The purpose of the parameter is to inform the network of the shape of the data it will be training.

The last layer should always contain the same amount of nodes as the number different outputs our system has. This assures that the output of the network will map directly to our classes.

A simple network consisting of three LSTM layers, three Dropout layers, two Dense layers and one activation layer. To calculate the loss for each iteration of the training we will be using categorical cross entropy since each of our outputs only belongs to a single class and we have more than two classes to work with. And to optimise our network we will use a RMSprop optimizer as it is usually a very good choice for recurrent neural networks.

The *model.fit()* function in Keras is used to train the network. The first parameter is the list of input sequences that we prepared earlier and the second is a list of their respective outputs. In our tutorial we are going to train the network for 200 epochs (iterations), with each batch that is propagated through the network containing 64 samples.

Generating Music

After training the network it is time to have some fun with the network we have spent hours training.

To be able to use the neural network to generate music you will have to put it into the same state as before. For simplicity we will reuse code from the training section to prepare the data and set up the network model in the same way as before. Except, that instead of training the network we load the weights that we saved during the training section into the model

Now we can use the trained model to start generating notes.

Since we have a full list of note sequences at our disposal we will pick a random index in the list as our starting point, this allows us to rerun the generation code without changing anything and get different results every time. However, If you wish to control the starting point simply replace the random function with a command line argument.

Conclusion and Future Work

We were able to show that a multi-layer LSTM, character-level language model applied to two separate data representations is capable of generating music that is at least comparable to sophisticated time series probability density techniques prevalent in the literature. We showed that our models were able to learn meaningful musical structure. This paper's writing comes at an interesting time in the space of deep learning generated art. In the last week, Google has announced its new Magenta program, a TensorFlow-backed machine learning platform for generating art. Google also released a 90-second clip of computer-generated melody with an accompanying drum line. Given the recent enthusiasm in machine learning inspired art, we hope to continue our work by introducing more complex models and data representations that effectively capture the underlying melodic structure. Furthermore, we feel that more work could be done in developing a better evaluation metric of the quality of a piece – only then will we be able to train models that are truly able to compose original music!

References

[1] https://deeplearning4j.org/lstm.htm

[2] https://keras.io/

[3] https://colah.github.io/posts/2015-08-UnderstandingLSTMs/

[4] Chun-Chi J. Chen and Risto Miikkulainen. Creating melodies with evolving recurrent neural networks. Proceedings of the 2001 International Joint Conference on Neural Networks, 2001.

[5] Douglas Eck and Jurgen Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical Report No. IDSIA-07-02, 2002.

[6] Daniel Johnson. Composing music with recurrent neural networks.

[7] K. Choi, G. Fazekas, and M. Sandler, “Text-based LSTM networks for Automatic Music Composition”, 1st Conference on Computer Simulation of Musical Creativity, 2016