Music Generation with LSTMs

Tobby Lie
Kevin Macfarlane
Devin Piner

***Abstract*-- Historically music has always been an integral component of human life and interaction. Music has always been something innate to people and has served a powerful role in conveying powerful abstract emotions that humans often times have a difficult time dissecting. At the same time, peeling back the layers of a musical piece reveals that it is afterall just a sequence of notes strung together with some constraints such as key and time measure. The problem statement and motivation provides insight into why this topic is being explored and what fruit it actually can bear. We then move on to the methods we implemented in order to solve our problem. What we hope to achieve by the end of this paper is, given sequences of musical pieces in musical instrument digital interface (MIDI) format, is it possible for a long short-term memory (LSTM) to be trained over several epochs to learn to generate a unique sequence of notes, we explore an LSTM implementation for both ABC-notation and midi formats for input into our network and discover interesting strengths of both methods. We utilize an LSTM because it must have the ability to learn past of its ability to remember from past time steps in the network. After describing our methods we move on to discussing our results both in the actual turnout of our audio generation as well as plots describing loss and accuracy of our model during training. Finally, we discuss related works and limitations in our last portion, which pertains to implementations and ideas within a similar scope that have been done as well as limitations to the general goal of utilizing LSTMs to generate sequences of notes.**

I. PROBLEM STATEMENT

How would it be possible to quantify something abstract? How could we generate new creative pieces of music with neural networks? Our aim is to determine, given existing pieces of music, how a neural network - specifically a recurrent neural network, could generate new music based off of what it has learned.

II. MOTIVATION

Music in itself is an interesting artform that brings people together and is the backbone of many other artistic mediums such as film or video games. It is often believed that creating music requires 'feeling', but what does this mean exactly? Can a neural network create meaningful and interesting content? Perhaps from the synthesization of sound based on what an LSTM has previously learned, it is possible that we can learn what makes something that is thought to be immeasurable, something that is indeed concrete and calculated, and perhaps from this we can apply what we learn to other interesting topics under the same umbrella or intersection of all things creative.

Aside from the initial novelty of training a network to create something that is at face value abstract, it was also of great interest to determine the effectiveness of an LSTM implementation with regards to generating sequences based off of existing sequences. We've seen in other applications text to text generation when given bodies of text to create an LSTM that can create a movie script with varying degrees of accuracy and believability[6]. It is highly intriguing seeing how something as mathematical as a neural network is able to model a concept that is often difficult for even humans to comprehend and dissect.

With any project, the primary motivation lies in the result and how satisfied we are with the outcome. In this project specifically, we define success in a somewhat abstract yet calculated way. We hope to generate a music that is both unique and believable for an audience. Unique in that we are able to synthesize sounds and sequences that are not exact replicas of the pieces the network is fed, such that music unique to our training data set is created. Believable meaning sound sequences that are deemed music, that are melodic, dynamic and pleasant sounding.

It will be of great intrigue to see how different the sequences generated will be when using the different forms of input, namely ABC-notation and midi. Will one yield more complex results than the other? This paper will quantitatively discuss the differences in utilizing the two separate formats. This will be studied through plots of performance, the differences in completion times of training, the quality of the sequences outputted by the model and various other metrics.

To reiterate the structure of this paper is as follows: Section III describes the methods utilized when approaching this problem, it will go through the neural network architecture as well as other pieces of code utilized in order to prepare data for training. Section IV goes in depth of the results derived from our model and discussion of observations made, specifically differences in approaches when utilizing ABC-notation versus midi. Section V explores related works within this scope of study to explore where it stands currently in the field and surely more developed and mature projects that are doing excellent work expanding on this sort of research. In section VI limitations are discussed and how far an LSTM can go in terms of generating passable music. Lastly, we talk about some takeaways and closing remarks about the project. We will include our github where the code is located in the references[5].

III. METHODS

Our goal was to take as input a data set of different pieces of music and as output generate a new piece of music that was coherent. For this the obvious choice of model was an LSTM neural network. LSTMs utilize gates in order to circumvent the vanishing gradient problem. An LSTM layer is an alternative method to compute the hidden state of a recurrent neural network (RNN).

A. Preprocessing and Data Preparation

Before going into detail of the architecture of our LSTM, let's talk about how we preprocessed our input data for training. One python3 library that was essential to this project was music21 which allowed us to manipulate and handle midi file data which was the format of our music files in our data set. This library allowed us to partition instruments, parse midi files, handle single notes and chords. We first created a function called get_notes which would get all the notes and chords from our midi files in a directory and then append them to a list called notes which the function returned. When we ran into a chord, we utilized normalOrder from music21 which is essentially an integer representation list of a chord[9]. After we get all of the notes, we then get the number of distinct pitch names.

We now move on to preprocess our sequences to be used with the neural network. Here we define a sequence of any length we choose. A dictionary is created that maps each note to an integer. We then create input sequences and output sequences. We do this by creating an input sequence of length that matches the sequence length and then create an output sequence which is the input sequence shifted right once in order to provide the next element for each element in the input sequence. After we have produced all of the inputs and outputs, we then return these in order to use them for training.

With the ABC-notation aspect, the process was similar. We would begin by getting the data from the text input file into a variable. We would then get all unique characters in the data string and create a character to index mapping of all the distinct characters. We then would do the reverse, getting the index to character mapping. From here we build our model, which will be discussed in the next section. After this we go through epoch by epoch and from a batch size we define, will train a certain number of batches determined by batch size each epoch and will from this creates a list of losses and accuracies. At every 10 epochs, the weights from the model will be saved so that they can be used if any sort of sequence generation needs to be done from a pre-trained model.

B. Neural Network Architecture and Training

Based on the number of unique pitch names and network input size we create our model. Our model consists of three LSTM layers, each followed by a dropout layer with given probability of 30%. Dropout is introduced after each LSTM layer in order to provide regularization in an effort to reduce overfitting[10]. We use 512 units in each of our LSTM layers. Units in this context are the neurons in each time step hidden layer of our neural network which should not be mistaken with timesteps which refers to how far back a model is able to remember its state. For reference, below this paragraph is an image of an LSTM unit. Following our last LSTM layer we have two Dense layers which are two fully connected layers utilized for outputting a prediction. Also important to note, we set the return_sequences argument to true in order to output an entire sequence and not a single value for each input so the next LSTM layer is able to have the correct input. We ultimately reach an activation layer which utilizes a softmax activation since we are dealing with a multi class classification problem in the last layer. We used categorical cross entropy as our loss function, and rmsprop as our optimizer. Finally we train our model for 200 epochs with batch size of 32 with about 16 midi songs in order to get some rather immediate results as more midi songs took much longer to train. We record our loss and accuracy for each epoch and plot our results as well as generated our new midi file which will be displayed and discussed in the next section.

Our architecture for the ABC-notation portion was very similar to the midi architecture, we utilized a first layer of embedding which is utilized to embed higher dimensional data into lower dimensional vector space [14]. We then used three LSTM layers each followed by a Dropout layer of 20%. This is followed by a TimeDistributed layer which is a wrapper that applies a layer to each piece of input. Finally at the end we utilize a softmax activation layer because we are dealing with a multi-class classification problem. It is also worth noting that our network was indeed many-to-many as we required a model that could deal with multiple parts of a sequence to be handled as input as well as a new sequence to be outputted. At each layer of the network we have 256 LSTM units and at the end have a number of output nodes equal to the number of classes. At every time step, the LSTM units generate the input into the next layer and also the same output will be used for the same LSTM unit. In Keras the return_sequences parameter is set to True in our case so that each LSTM unit will create output as the next character is given input from the previous character in the sequence[4].

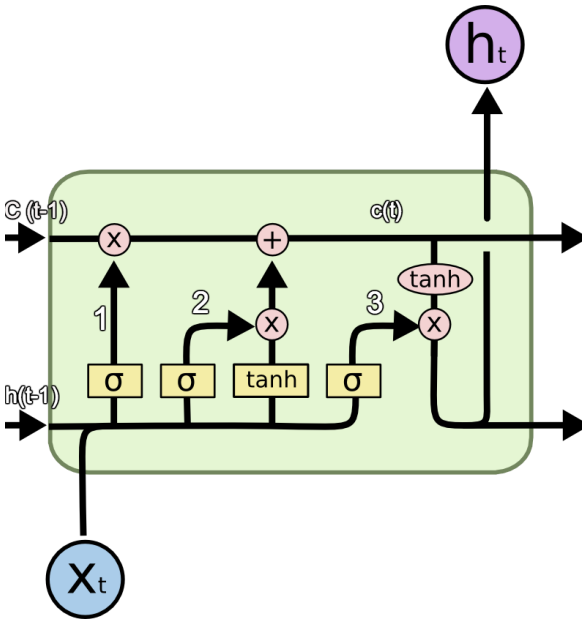

Figure 1: An LSTM Unit
(Source:http://colah.github.io/posts/2015-08-Understanding-LSTMs)

IV. EXPERIMENTS AND RESULTS

First let's discuss the generation of a new midi file given our trained model from the previous section. From our trained model, we were able to save the hdf5 file from each epoch executed within training. This way, we are able to reconstruct our model when needed for generation by just loading our saved weights to a model of the same architecture as used when training[1]. By utilizing our trained model we are able to initialize a random starting point within a range of 0 to the length of the network input to simulate starting at a random point in a sequence to begin predicting on[1]. We are able to define how many notes we would like to generate and in a for loop, go from 0 to our size of generated notes to essentially predict the next note in a sequence given some input sequence. We take the max value from our prediction vector and append that result to our prediction output list. Once we have our prediction output list, we can now generate midi from this list. For each element in our prediction output list we loop through and see if the current element is a chord or a note, as this will be handled differently by music21[9]. Once we have determined and parsed our note or chord, we can then append this result to a list of midi formatted sounds. After this we create a single midi stream with all of our notes and chords and write this to a file which we can import into a Digital Audio Workstation, to be played. The result is included below for listening.

We have also included our training accuracy and loss plot to display how our model performed over a span of 200 epochs. We utilized this plot in order to verify that our loss was in fact decreasing and that our model was performing well.

Result:
https://soundcloud.com/tobby-lie/rnn-trained-on-200-epochs-and-16-songs

Figure 2 below is a plot representing the training accuracy and loss for the model trained on 200 epochs. We can see here that the initial accuracy begins near 0.1 and initial loss begins at near 4.9.The initial accuracy increases within the first 25 epochs quite sharply and then starts to plateau after maintaining a slower increase at around 1.0. The opposite can be seen of the loss, it seems to sharply decrease within the first 25 epochs and then plateaus, decreasing much slower as it nears 0. The loss decreases smoothly without any spikes and the inverse occurs for accuracy.

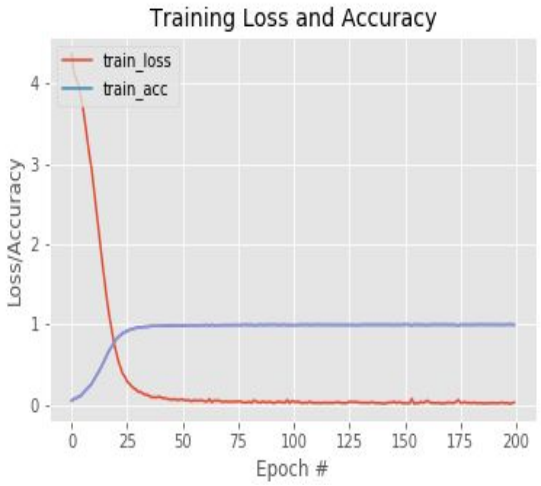

Figure 2: Training Loss/Accuracy vs Epoch Plot for MIDI

In the beginning of our project exploration we initially looked at an implementation of music generation via LSTM that utilized ABC-notation. ABC-notation consists of two parts, the first part is the meta data, which includes various kinds of information. X: indicates the tune, T: the title, M: the time signature, L: note length, R: type of tune, and K: key. Part 2 is the actual sequence of notes represented by characters[4]. Although easier than midi to understand and parse programmatically, this representation proved to be too simple and yielded too linear of results, ultimately providing rather boring music. The dataset consisted of ABC-notation represented in text file format.

The architecture of the model was very similar to our initial architecture with the exception of no Dense layers and just one final softmax activation layer. A result of one of the pieces will be provided below as well as another plot for the training of this model. The generated output from this

program was a text file that contained ABC-notation which we plugged into a program called EasyABC which played the composition from the text file and also provided the ability to export to midi format.

Result:
https://soundcloud.com/tobby-lie/rnnabc-80-epochs

Figure 3 below is a plot representing the training accuracy and loss for the model trained on 80 epochs. We can see here that the initial accuracy begins near 0.1 and initial loss begins near 1.6. The initial accuracy increases within the first 30 epochs and begins to plateau after this while the loss behaves similarly and inversely.

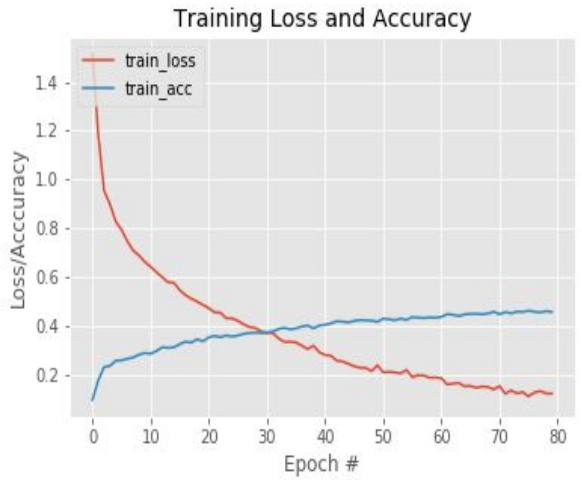

Figure 3: Training Loss/Accuracy vs Epoch Plot for ABC-Notation

In the initial experimental phase of this project, we first trained on the ABC-Notation sequences on a Macbook Pro 2019 and this was a very slow process. For a single epoch to run, it took at least 4 minutes with a sequence size of 100. This isn't the worst runtime in the world, however we knew that moving forward when dealing with more complex data and much larger data sets, there had to be a better way to implement training. We then moved our training onto Google Colaboratory which is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud[13].Google Colaboratory is an excellent free option for those in need of GPU computing resources. After running our initial training in Colaboratory with GPU resources enabled, training sped up to only a couple seconds per epoch. We tested our training on the midi data with various data input amounts and got various epoch runtimes. Without GPU enabled, it took over an hour to train each epoch with about 60 midi files fed into it. When trained in Colaboratory, this execution time reduced to about 10 minutes per epoch. As time was a limited resource with training, we decided to strip our data set input down to only 16 midi files which reduced each epoch with GPU enabled to about 73 seconds per epoch.

An issue we faced early on in the training process was that our network would output sequences that were just the same note played over and over again without even a change in time. It would be an equally sized note playing consecutively with no pauses. We realized that this was due to the fact that we needed to train the model for much longer as we were only training the model for about 25 epochs and the loss would not decrease enough for the model to learn anything useful from the dataset. After we bumped up the epochs to at least 100 epochs, it became apparent that we were now dealing with more complex sequence structures that were not anymore monotonic. We did notice however, that we were not getting any chords played with our sequences when utilizing ABC-notation format. As soon as we shifted over to midi format we noticed chords were included in the output sequences as well as more dynamic timing. Before we shifted to using 16 songs in our data set we tried it with just 4 midi files, which outputted very similar sounding sequences telling us that the network probably needed an increase in dataset size which is why we increased the dataset to 16 midi songs. This is seemingly improved the variance and uniqueness of our network output and when trained for 200 epochs, we were able to achieve around 0.02 loss which we found to be a sufficient stopping point for our training. It seems that our loss would continue to decrease if we let the model train for even longer, but for the purposes of testing for a solution, 200 epochs served that purpose.

In order to get our midi files into a format that was listenable, we utilized LMMS a lightweight digital audio workstation (DAW) in order to simply import the midi file into the project and slap on any sort of sound or instrument provided by a free library. We utilized Freeboy, a Gameboy type sound emulator that provided a retro video game soundtrack type of sound. The benefit of utilizing midi as the dataset format is that we get the sound sequence sounding however we want with the aid of any DAW such as Pro Tools, Logic Pro, Garageband and many others. This makes it easy to test how varied and rich our outputs are from the network, not to mention midi files are abundant as it is a standard in the music industry.

V. RELATED WORKS

Clara

A long short-term memory (LSTM) neural network that composes piano and chamber music. She is based on the idea that you can generate music in the same way that you can generate language. For example, when training a language model, we feed the neural network a paragraph and then ask it to predict the next word ("to be or not to _____").

Once a model is good at predicting the next word, you can turn that into a generator. As it predicts the next word, you feed that word back into the model and ask it to predict the next word after that and so forth. A huge portion of this project was finding a good encoding for them music[11].

Magenta

Developed by Google Brain, it was aimed at creating a tool for artists to use when working on new songs. At the end of 2016, they published an LSTM model tuned with Reinforcement Learning. The interesting idea was to use Reinforcement Learning to teach the model to follow certain rules, while still allowing it to retain information learned from the data. With certain metrics defined, a "music theory rule" had been defined. This was used to determine the extent of the reward given for a particular behavior. More recently, the Magenta team has used GAN and Transformers to generate music with improved long-term structure[12].

VI. LIMITATIONS

Although our LSTM architecture provided some coherent and pleasant musical sounds there are limitations to producing musical pieces in this way. Given several musical sequences, our model was able to learn from this information and synthesize new sequences based upon what it has been trained on. Using this method allowed the neural network to be trained on and understand the structure of the sequences it was provided and was able to create new solid pieces that made sense musically and were passable as musical pieces.

When we take a step back and listen to entire pieces generated by the network, it seemed that the limitations became more clear. The LSTM model was able to create the next element in a sequence by looking back and from the softmax choose the most likely element to be next in the sequence. This created positive results but with limitation. There would occasionally be notes that were repeated if not trained enough, for example when we trained a model with loss of 0.5 or greater, it seemed to just create repeating notes that were identical and of course this isn't really music or at least complex music. Another observation that we made was of course there wasn't any true cohesive structure throughout the entire piece as a whole, the network essentially spits out a nice sounding sequence but with no overall integrity and global identity. There is no sense of full structure and predictability which is what traditional music structure provides. Proper music for the most part has some sort of pattern and what we generated lacked that, therefore creating sounds that were confusing and seemed to wander with no sense of groundedness. If our pieces were given to an actual musician, I'm sure they would be able to detect flaws and limitations. Specifically in regards to seeking some cohesive structure that ties the entire piece together. Issues would arise within pauses and dynamics as the pieces are very constantly paced with no use of pauses or sense of dynamics. These issues would make it obvious to any musician and even non musician that there is a sense of synthetic and non-organic to the generated pieces.

Another notable limitation for this project was time. In the beginning, we tried to train the network on about 30 midi files and this was estimated to take two days to train which is not ideal when we want to constantly tinker and play around with our training to get different results. We eventually reduced our number of midi files to below 15 which did help with our training time, as well as utilizing Google Colab's gpu in order to accelerate the process but even then training took several hours. The amount of time to train a model for this project was significant and contributed to the difficulty in deriving results to analyze.

VII. CLOSING REMARKS

The exploration of RNNs in the pursuit of music generation has led us to an excellent place. We started with testing an LSTM with simple ABC-notation compositions and from that we generate simple monotonic sequences which, even though very simple were melodic and in key. We then moved onto using an LSTM based network to train a network to learn from midi files. Out of the second implementation, we were able to produce more varied and dynamic sound sequences that also included chords which definitely produced more interesting rich sounds. From our midi implementation, we have been able to create a general framework in which our network can learn from a plethora of widely available midi files online. We specifically used Final Fantasy midi tracks for testing but with the flexibility and accessibility of midi, we could easily adapt to other genres of music. If given more computing power and time, it is very curious to discover how much better our model would be able to perform if given a much larger dataset with varied genres. In this case we may need to tinker with our model more to try and get better performance.

REFERENCES

[1] Skúli, Sigurður. “How to Generate Music Using a LSTM Neural Network in Keras.” *Medium*, Towards Data Science, 9 Dec. 2017, [Online]. Available:https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. [Accessed 15 November 2019].

[2] Mangal, Sanidhya, Rahul Modak, and Poorva Joshi. “LSTM Based Music Generation System.” IARJSET 6.5 (2019): 47–54. Crossref. Web.

[3] Kotecha, Young. “Generating Music using an LSTM Network.” arXiv (2018): 1-8. Crossref. Web.

[4] Sharma, Gaurav. “Music Generation Using Deep Learning.” *Medium*, Data Driven Investor, 25 Jan. 2019, [Online]. https://medium.com/datadriveninvestor/music-generation-using-deep-learning-85010fb982e2. [Accessed 15 November 2019].

[5]Project Repository: https://github.com/tobby-lie/Multi-Instrument-RNN-Generation

[6] Newitz, Annalee. “Movie Written by Algorithm Turns out to Be Hilarious and Intense.” *Ars Technica*, 9 June 2016, [Online]. https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/. [Accessed 15 November 2019].

[7] MIDI data: https://github.com/Skuldur/Classical-Piano-Composer/tree/master/midi_songs

[8] ABC-notation Data: http://abc.sourceforge.net/NMD/

[9] Music21 documentation: https://web.mit.edu/music21/

[10] Budhiraja, Amar. “Learning Less to Learn Better - Dropout in (Deep) Machine Learning.” *Medium*, Medium, 6 Mar. 2018, [Online] https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. [Accessed 15 November 2019]

[11] Harrington, Charlie. “Generating Classical Music with Neural Networks.” *FloydHub Blog*, FloydHub Blog, 15 Aug. 2019, [Online]. https://blog.floydhub.com/generating-classical-music-with-neural-networks/. [Accessed 15 November 2019].

[12] Spezzatti, Andy. “Neural Networks for Music Generation.” *Medium*, Towards Data Science, 24 June 2019, [Online]. https://towardsdatascience.com/neural-networks-for-music-generation-97c983b50204. [Accessed 15 November 2019].

[13] Google Colaboratory: https://research.google.com/colaboratory/faq.html

[14] rajmehra03. “A Detailed Explanation of Keras Embedding Layer.” *Kaggle*, Kaggle, 6 Jan. 2019, [Online]. https://www.kaggle.com/rajmehra03/a-detailed-explanation-of-keras-embedding-layer. [Accessed 15 November 2019].