EE456 Final Project: Neural Network MIDI Creation

James Connelly

December 12, 2023

1 Abstract

This project consists of two different types of recurrent neural networks: a Long short-term memory (LSTM) network and a Gated Recurrent Unit (GRU) network. The project's main goal was to produce enjoyable music from a neural network which was given a large dataset of piano classical music pieces. The reason two networks was used was to compare the various ways that networks can be trained to learn about music. Music is a somewhat free form style of data in that there is not necessarily one specific desired input or one specific output inherent to the data. Consequently, those decisions had to be made independently and therefore I thought it would be interesting to compare two different styles of network and see how they produce notes.

The LSTM network was given note data in the form of vectors describing pitch, duration, and step (time between notes). Then, it would predict what it thinks the next note should be based on the prior notes given to it. This produced some interesting melodies but ultimately, did not provide the musicality that I was hoping for.

The GRU network is similar in structure to an LSTM network but it was given note data in the form of a large text file at which point it analyzed the text and tried to recognize patterns of the text instead of vectors. Its output was text in the form of a midi file that could then be played so this approach was more of a linguistics problem than a math problem.

The outputs are not necessarily pleasing to listen to, but they do illustrate a proof of concept for the RNN structures which was important step in the right direction for a novice neural network builder.

2 LSTM Network

2.1 Outline

LSTM networks are very popular for music generation [8, 9]. Simple recurrent neural networks (RNN) can do some sequential input analysis, but the LSTM has the forget gate which allows for more long term memory. For music, long term memory is important because songs are often repetitive and the context of future notes is decided by what was played previously. Additionally, this network utilizes dropout layers which should allow for faster network training and less risk of overfitting.

The structure and architecture for this network was original in some parts and largely samples the TensorFlow manual in order parts [4]. This aided me in network creation as I do not have much prior experience with building neural networks from the ground up. The sample code allowed for me learn what I was doing without wasting too much time debugging endless problems. I will elaborate on this in subsequent sections, but I made sure that libraries/tools/code that I borrowed were only used if I could understand what they were doing which influenced some of my network design choices [13]. I thought this was the most authentic way to learn from the project even if I can't come up with all the code on my own.

The purpose for this project was to investigate how neural networks can achieve "creativity." When AI started to boom in the past 5 to 10 years, the creative fields like art, music, and conversation, were some of the areas that which I thought would take AI a long time to catch up to humans. However, tools like Chat-GPT and Dall-e prove that domains which appear creative are actually formulaic. There are various examples of music generating AI like MuseNet or Jukebox by OpenAI [6, 7]. My goal for this project was simply to learn how networks go about achieving "creativity" and try to generate my own musical outputs.

2.2 MIDI Dataset Creation

The first task was to generate a dataset. Luckily, there are many midi datasets that are already in existance. For this project I utilized the MAESTRO dataset V3.0.0 [2]. This dataset provides over 1000 midi files of mostly classical piano pieces which was more than enough to train the network on. A key aspect of this dataset is that each midi file only uses one midi instrument which greatly simplifies the burden of the network to recognize patterns. One consideration that I had to have early on was whether I wanted to use music in the form of a wave or a more quantized midi format. I chose to go with the midi format because neural networks that deal with sound in wave form tend to be subject to a lot more noise and other errors which I thought would be easy to eliminate with the mathematical simplicity of midi files.

However, midi files were still not easy to deal with. In order to convert the midi file to a more usable format, I used the Pretty_MIDI python library which allows the midi file to be stored in an array format. At this point, the MIDI file is converted into a Pandas dataframe (a common library for neural networks) with the following features: pitch, note start time, note end time, note step (time between note and previous note), and duration (note end time minus note start time). This conversion is completed for as many files as are needed to train. When training this network, I only used around 30 of the 1276 files which is a small sample size but it did make the training time upwards of a few hours to go any higher which was unrealistic for the tests that I needed to do.

	pitch	start	end	step	duration
0	74.0	1.023438	1.085938	0.000000	0.062500
1	81.0	1.104167	1.153646	0.002604	0.046875
2	48.0	1.209635	1.217448	0.005208	0.045573
3	64.0	1.102865	1.154948	0.000000	0.055990
4	76.0	1.179688	1.207083	0.003906	0.054688
..	...	...	...	...	...
404	48.0	147.998698	148.069010	0.002604	0.066406
405	57.0	148.019531	148.071615	0.141927	0.028646
406	45.0	148.204427	148.266927	0.000000	0.040365
407	51.0	142.904427	143.053385	0.178385	0.049479
408	77.0	142.949219	143.095052	0.239583	0.054688

Figure 1: Dataframe Example

The next step of the data creation was to build an actual Tensorflow dataset from the Pandas frame. All this required was condensing down the frame by removing the start and end columns which are not necessary to train the network and then create sequences of data such that the input data will be a sequence of notes and then the target value will be the next note in the sequence. That means that the network is being trained to predict the next note in a sequence given a list of input notes which we will see again in the evaluation section.

2.3 Network Structure

The network structure consists of an LSTM layer, a dropout layer, an LSTM layer, a dropout layer and then 3 dense output layers that give the pitch, step, and duration for the output note.

Each LSTM layer has the structure provided in the figure. There is the forget gate, input gate, output gate, and candidate memory. The forget gate does exactly what it sounds like as it essentially deletes unnecessary information from the memory. The input gate takes the input and can add new information to the memory. The output gate decides what to output from the network given the current network. Finally, the candidate memory creates information that will be later added to the memory. In the keras library (a part of the Tensorflow library), the LSTM structure uses the typical hyperbolic tangent activation function by default [10].

After each LSTM layer, there is a dropout layer. Based on other research papers, networks that used dropout seemed to give results that I thought were satisfactory so I decided to include dropout [11]. The dropout functionality means that during the training epochs, some neurons are deactivated to avoid fragile dependencies and overfitting. This is especially important for the output of this network. I did not want the network to be overfit because then it could just generate a "song" which already exists in the dataset which does not really constitute my definition of creativity. I set the layer to deactivate 20% of the input neurons to the layer each time which seemed to be a good number for run time.

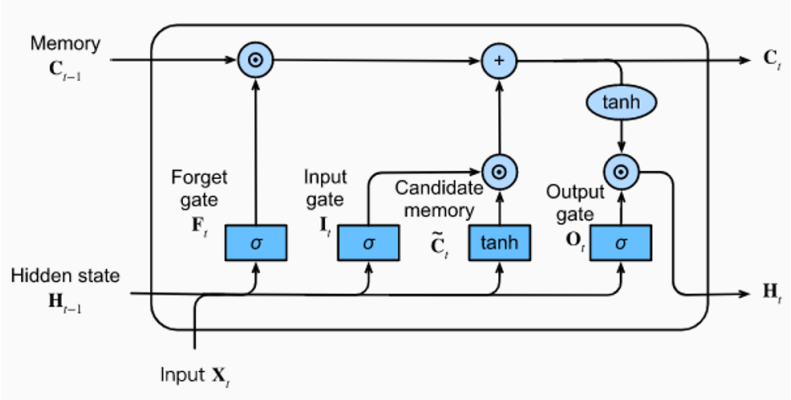

Figure 2: Sample LSTM Cell [10]

Where did this network structure come from? During my research, I found one project called deepjazz.io which used a similar network structure to generate new jazz music [12]. Based on that network's outputs I thought it would be a good idea to try and base my own upon it.

2.4 Loss Functions

Loss functions are a tricky thing to develop for a network like this. Based on the input data and labels, there has to be decisions made as to what the network should be learning. For this network, it is learning to predict the next note of the midi. While I originally thought this was a sufficient method to create interesting melodies it may be a little bit too broad of a task as the outputs show. This also helped spur my interest in making the second network.

Ultimately, spare categorical cross entropy was used for the pitch's loss function. This is typically written in the following form:

$$\text{Loss} = -\sum_{i=1} nt_i \log(p_i) \tag{1}$$

In this case, n is the number of classes we are trying to represent (128 notes in this case) and p_i is the softmax probability for the class i. This function is often said to work well for outputs that are not one-hot encoded vectors which is good for this network which is outputting a probability of the most likely note to choose.

For duration and pitch, a Mean-Squared-Error loss function was used because we just want to bring those numbers closer and closer together. However, it was recommended that a positive pressure is added which

means that follows are forced to be more positive to avoid negative pitch and duration outputs which do not make sense for this network.

2.5 Training

The network was trained with the following parameters:

Input Files:	30
Epochs:	30 (stopped at:)
Learning Rate:	0.005
Batch Size:	64
Sequence Length:	100

Table 1: Various Parameters Chosen for Model Training

These parameters were chosen through mostly just iterative testing. It was hard to hear much of a difference when changing the learning rate unless it was changed drastically which often made the output unusable. Batch size influences the number of training samples the network sees in each epoch. 64 made it so that the network did not run forever but still gave interesting output. Sequence length is an interesting parameter. It basically changes how much data is fed to the network at once. The higher the sequence length means more of a temporal connection can be made but making it longer also significantly slowed down the network so 100 had to be used.

In terms of stopping condition, I decided not to use a validation set to stop the training of the network and instead just use the loss function as a stopping method. If the loss function increases too many times in a row then the network would stop training. The network does not generalize particularly well to new input data because music is unpredictable and while context can give it a good guess as to what note should be played next, I realized it will never do well in the validation step because it cannot know for certain that a certain composer will choose C or D for the next note in their piece. However, the network's purpose is not to generalize because it is meant to generate it's own unique output based on patterns it has seen so this is not a big problem but just an observation that I had.

2.6 Evaluation

In order to evaluate the network, the network weights were saved at the end of each epoch into a folder. Then, once the network was done training, I ran all the networks weights through their own generative process so that it could be observed how each epoch generates its own output.

The generative process worked like this: first, a sequence of sample notes was obtained. I decided to just grab the first sequence of notes from the last midi file in the dataset since I knew that the network hadn't been trained on it. Then, the next node is predicted using the keras predict function which just takes the input I generated and then creates an output. Additionally, there is a parameter called temperature which scales the outputted pitch to provide some randomness to the output so that the output does not get stuck one note. This was recommended by Tensorflow as a common problem with RNN and overall, it seemed to give the best results when temperature was around 2.

The generated notes are then fed back into the Pretty_MIDI python library at which point a MIDI file is output which can be listened to.

2.7 Results

The results are hard to show visually, but there are a few graphs to show. The midi outputs for this network can be found in the net1_midi folder of the ZIP file. It has a midi file for each epoch of the network while it was training so that one can observe how the output changed over the training.

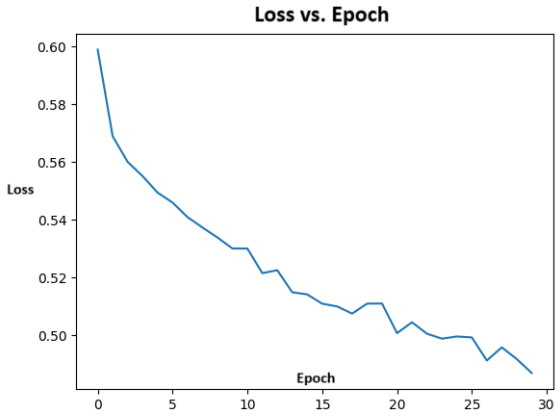

Figure 3: Loss Function Result for the LSTM network

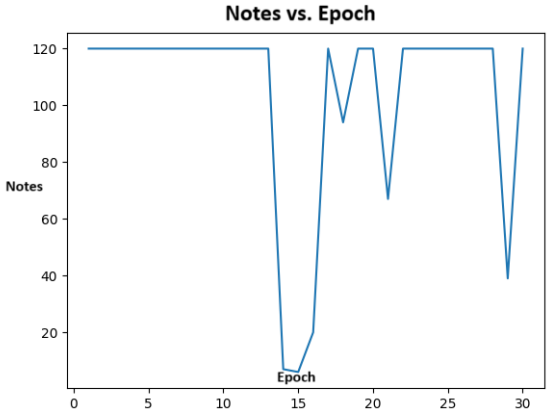

Figure 4: Notes generated per epoch of each of the midi files generated

The second graph came about from a problem that I noticed while training the network which was that, even though I was requiring 120 notes to be predicted, the network would not always produce 120 notes. Sometimes, it would make more and more notes have 0 duration as it trained more and more epochs. That problem seems to have been mostly mitigated in this final run but it was an interesting problem. I predict it has something to do with overfitting.

As for the sound of the output, I would say that it never really produces a pleasing output. It seems to have had a hard time differentiating the bass line from the melody. I noticed that in many epochs it oscillates quickly from playing a low note to high note which is a problem the input being for the piano. Pianos typically have two parts being played, one with each hand. As a result, the network seems to have vaguely picked up on that but it cannot combine those to make a more natural sounding line.

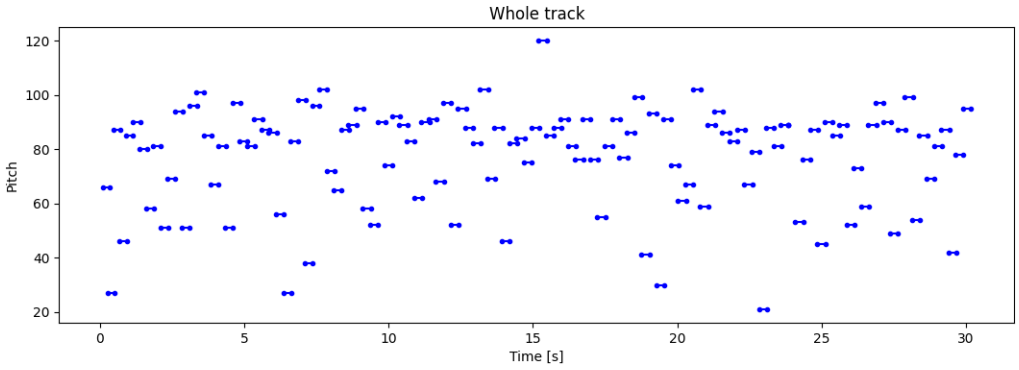

Figure 5: Piano Roll of Midi Output from Epoch 3

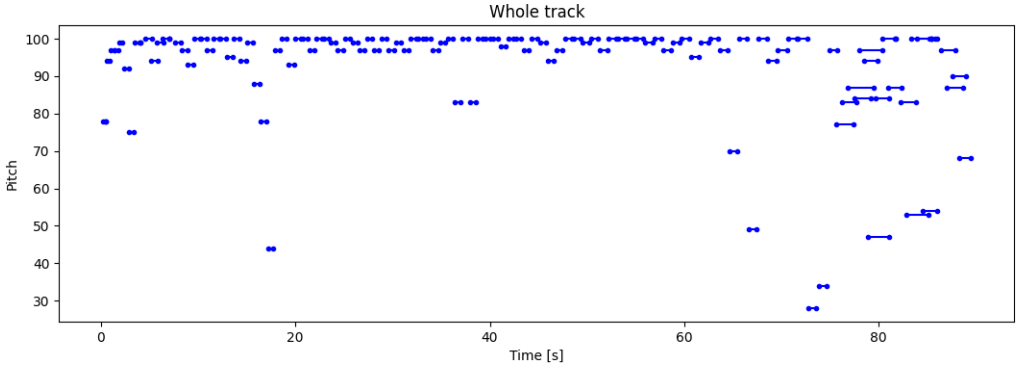

Figure 6: Piano Roll of Midi Output from Epoch 30

These two images provide an interesting analysis of what happens as the network trains. The network begins by producing notes all over the pitch possibilities but then consolidates near the highest pitches as it keeps training. I am not one hundred percent positive as to why that would be happening other than the fact the melody notes are dominating the network loss function since they are played fast and frequently so it is often the next note that needs to be predicted while the slower bass notes lag behind and result in this uneven distribution.

The output also got much longer at epoch 30 which is another interesting observation. This made me realize that my observation about less notes being played was really just the fact that the notes were being spread out more slowly most likely.

3 GRU Network

3.1 Outline

After receiving the first result, I was not satisfied with my output so I decided to try a different type of network to see if the results could be better. This network is still an RNN but it uses gated recurrent units (GRU) to predict the output based on the sequence. Again, TensorFlow and keras were used to construct the network.

I chose this network because it works well with text-based inputs. The text-based idea came from a YouTube video that did a similar a project with generating music [1]. For that project, the creator used a text based input and then an RNN network developed by Andrej Karpathy [**Karpath˙2015**]. This RNN has been cited for many different uses and he gives examples on it training on data from Shakespeare to baby names. What I liked about this network structure is that it appeared to have a higher chance of picking up on some ideas of harmony compared to the previous structure. At first, I tried to use Karpathy's RNN because of all its accolades but it only seems to be able to run on Linux so I switched to a TensorFlow example instead.

3.2 Input Formatting

For this network, I needed the input to be a large text file. I initially tried to format my midi in to text like the YouTube video that I watched but that would not work because the midi files in that video were formatted differently than the MAESTRO dataset (the way they stopped notes from playing was different), and the MAESTRO dataset was more difficult to deal with. However, I noticed from the previous network that the Pretty_MIDI library produced a nice line-by-line string structure for the midi file that I could use.

So, I converted all of the 1276 MAESTRO midis into one massive .txt file which gave a file size of 300Mb. This was way too large and would've caused the network to take months if not years to train. I reduced the number of files to just 10 which was a few Mb and within the recommended size.

Next, the characters were further manipulated into a mapping done by TensorFlow's StringLookup method which essentially just encodes each character to a number. This would matter more if the string input wasn't already numbers but I did it anyways because of decimal points and escape characters. Sequences were created much like the last network where a sequence of 100 was again used so that the network could predict what the next character would be. Finally, the dataset was ready to go into the network.

3.3 Network Structure

The network structure is more simplistic than the previous LSTM structure. Essentially, it is just an embedding layer, a GRU layer, and then a dense output layer which produces probabilities for what the next character should be [5].

The network takes in a 100 sequence length input into an embedding layer. From what I have gathered, the embedding layer maps similar words to similar vector space. Then, the GRU is just a simplified LSTM with two gates: a reset gate and an update gate. Finally, the GRU outputs are matched to a probability where the new character is chosen.

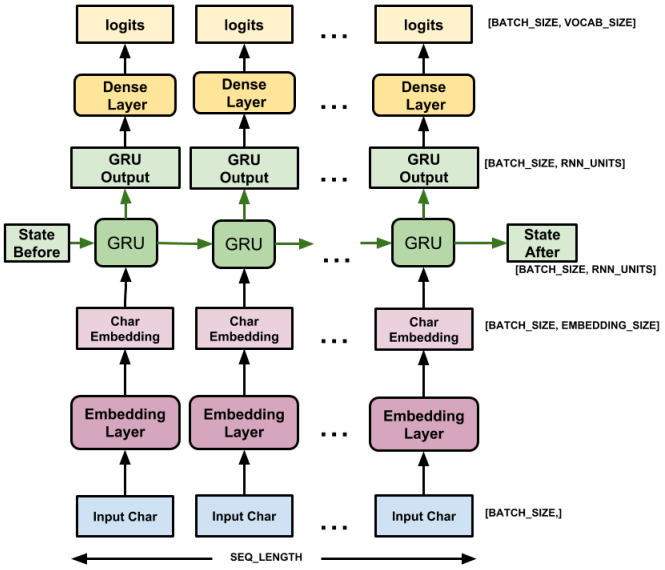

Figure 7: Network Structure [5]

3.4 Evaluation

Evaluation was difficult for this network. First, a prediction method was completed in a similar way to the LSTM. The network was primed with a sample input and then the rest of the output was generated. From there, a text file was created with the text in the format of a Pandas Dataframe. The text was manipulated back into a dataframe and then fed through a function which converts the dataframe back into a midi file at which point the midi file could be played.

3.5 Results

The results for this network are similar to the last. Overall, I was impressed by how well the network could emulate the style of input it was given. It easily picked up on the 5 column format and some of the rules between columns where the start should always be less than the end or the duration and step should always be positive. However, the actual musical output left a lot to be desired. It sort of just produces a random slew of notes albeit with a bit more of a chord structure than before which is what I wanted to happen.

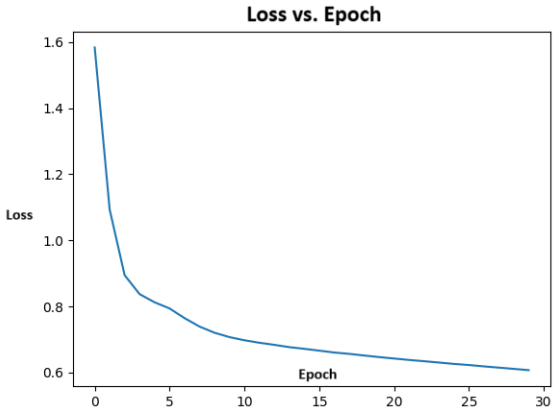

Figure 8: Loss Graph for the GRU network

The output can be listened to in the net2_out folder in this ZIP file. Even though I kept checkpoints for all the epochs' weights, I did not build outputs for all of them because the early ones were not formatted well enough by the network to be properly translated to midi file so there is just the one output. Either way, it is enough to see that this network structure is capable of learning the patterns and I would argue even better than the first seeing how the loss function drops so quickly.

This structure of data may actually be an easier way to pick up patterns in music. I would consider this string input a more "human" way to interpret music in that we are picking up on the patterns of consecutive notes and their lengths in a text format much like a piece of sheet music so maybe that is why the network is so good at picking up on the format quickly even if it could not pick up on how to develop a good melody or some harmony.

4 Difficulties

This project came with a lot of difficulties which ultimately led to some not great outputs. First, dealing with midi files was much more difficult than what I expected. Midi files are claimed to be this great computerized music style and while for some uses they definitely work well, it was difficult for me to wrap my head around formatting everything properly so that they could be used to train a network.

Actually, my initial idea for this project came from a YouTube video about building an Autoencoder network to produce musical output [3]. However, after researching that network for a few days, I just could not wrap my head around how the data was being formatted and fed in to the network so I decided to not go down that route so I could actually build something that I understand how it works even if the code came from a tutorial.

4.1 The Dataset

Another aspect that I have been thinking about a lot is the dataset. I am not sure if classical music is the write dataset to train on as an easy way to produce pleasing musical outputs. While classical music did lay the groundwork for much of the music theory utilized today, it usually does that in a more complicated

and less repetitive way than rock or pop music. I would like to see how these networks perform on different datasets. The Autoencoder network that was previously mentioned was trained on video game music midi files and produced from very nice outputs which makes sense considering the repetitiveness of music in games for aspects like menu or level music that just drone on and on.

Sadly, those datasets are hard to come by and again, difficult to process, especially if they include multiple instruments which was something I was trying to avoid.

4.2 Harmony Generation

For the first network, it makes sense that harmony cannot be generated. It is being trained to only really pick the next singular note, so it does not care how that note sounds in comparison to notes being played at the same time it only cares about how it compares to the previous notes.

For the second network, I think it could get a lot closer to learning harmony because when it sees the input, it is seeing the sequence size of inputs and notes that are close to each other in the midi string format are likely meant to be played at the same time and are therefore related. It does not seem to have picked up on that relation much but maybe with a bigger sequence size and more training time it could do that. I wanted to try this, but making the sequence size larger made the training time astronomically longer which was not realistic for this project.

5 Conclusions

Overall, the results are promising but certainly not ideal. The generated midi files are not necessarily pleasing but still interesting to listen to.

In comparing the two networks, the LSTM produced a more constant melody while the GRU produced a more sporadic sequence of "chords." It was good to see that output because it reinforced the ideas I had in my head about their respective loss functions and network structures. While they both take in a sequence size of inputs and use that to predict the next output, the LSTM was really good at predicting just the next note while the GRU could more build a sequence of notes together since it only predicted one character of a sequence at a time and had many opportunities to group these characters together as it built each line.

For takeaways, I think what I was able to get a lot from this. I learned a lot about the two different structures of RNNs. I understand both LSTMs and GRUs more than I did before. Also, since I used this built-in libraries, I was able to see some example code of how these networks could be built which was really helpful for me because I have very little experience with coding neural networks. I tried to come up code myself but as someone who has not done a lot of work with neural networks or midi files. I needed assistance in making the two communicate properly.

I also learned a lot about the importance of data for neural networks. I think this is something I had not seen before this project because all the other projects during the semester came with their own, nicely formatted, datasets. Trying to deal with the midi files was a nightmare and forced me to cross a lot of interesting ideas off of my list of network structures because I could not figure out how to make the midi files look a certain way without losing too much information in the process. Also, it is important to consider the features that were present in the dataset I provided and whether or not they were really the features I wanted to be reproduced (repetitiveness vs. uniqueness).

What does the network really accomplish? Ultimately, I was given two choices: overfit the network or let it be "creative" on its own. I tried to make sure the network did not overfit and just produce the exact melodies it was given because that takes the purpose out of the entire project. We do not need neural networks that can produce data we already are aware of. However, on the end of the spectrum that I ended up, it seems that the network was underfit to the point where it did not really produce meaningful output. Nonetheless, it was interesting to see any output at all and I would argue, who are we to say that the network's output is not musical? Maybe it is just innovating to a new genre of avant-garde music that has never been seen before and this network will be the impetus to the greatest musical revolution in the history of the universe... or maybe not.

References

[1] YouTube, Mar. 2017. URL: https://www.youtube.com/watch?v=SacogDL_4JU&ab_channel=carykh.

[2] Oct. 2018. URL: https://magenta.tensorflow.org/datasets/maestro.

[3] YouTube, July 2018. URL: https://www.youtube.com/watch?v=UWxfnNXlVy8&t&ab_channel=CodeParade.

[4] Oct. 2023. URL: https://www.tensorflow.org/tutorials/audio/music_generation#create_and_train_the_model.

[5] Nov. 2023. URL: https://www.tensorflow.org/text/tutorials/text_generation.

[6] URL: https://openai.com/research/musenet.

[7] URL: https://openai.com/research/jukebox.

[8] Raphael Abbou. *DeepClassic: Music Generation with Neural Neural Networks*. Stanford Univeristy, 2020.

[9] Shikhar Bhardwaj et al. "International Conference AUTOMATICS AND INFORMATICS'2022". In: *Automated Music Generation using Deep Learning*. IEEE, 2022, pp. 193–198.

[10] Ottavio Calzone. *An intuitive explanation of LSTM*. Apr. 2022. URL: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

[11] Daniel D Johnson. *Composing music with recurrent neural networks*. Aug. 2015. URL: https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks/.

[12] Ji-Sung Kim. *Deep Learning Driven Jazz Generation*. URL: https://deepjazz.io/.

[13] Keras Team. *Keras Documentation: LSTM Layer*. URL: https://keras.io/api/layers/recurrent_layers/lstm/.