Digital DJ

Cannon Lock
clock@wisc.edu

Prithvi Tanna
ptanna@wisc.edu

Nicole Athanasiou
nathanasiou@wisc.edu

Abstract

Several attempts have been made at trying to reproduce human made music using deep learning. Despite these attempts, all models seem to fall short of perfection when it comes to capturing the nuances present in music. In our report, we decide to remove some of these components of music and instead focus on some of the key ones. Using deep learning, we will extract the features of 1800 drum music files. These basic features can be utilized to create music of our own. The model will use a combination of techniques from models used from both natural language processing and other music based deep learning methods. Through the use of a Recurrent Neural Network and more specifically a LSTM network, we will break musical pieces into their components and then put them back together trying to mimic those originally made by human artists. The results of the model demonstrate that rhythm and pitch can be extracted from the drum beats but also exemplifies many avenues of improvement. Some of the key areas of success that we found were the ability to mimic rhythm and timing, but in the end the model fell short in its ability to pick the correct pitches. We found that deep learning can be used to create musical pieces that mimic those created by humans when broken down into some of its key components.

1. Introduction

Music is vital to culture worldwide. Musical composers can spend a lifetime learning how to produce music efficiently, and we would like to help them speed up some aspects of the process. In a society that praises musicians constantly, talent is often associated with music. As researchers in search of the same praise we wish to create a deep learning model that can generate music that is passable as human-made. Perhaps our Digital DJ can keep up with these trends by producing fresh music. We do not hope to take all the jobs of DJ's around the world, but instead hope to supplement their set of background beats and tracks for them to pull from. Using deep learning we hope to cut down on the amount of time they have to spend creating simple beats and focus on some of the smaller facets of this art that deep learning can capture.

There has already been research done on deep neural networks for music generation [5]. Using a variety of inputs from music tabulation to digital files these models have attempted to capture the elements of the music they trained on. Known challenges to approaching music composition using deep learning include lack of musical rhythm, structure and melody [1] in the pieces that are created. We will attempt to address a subset of these challenges throughout our project to demonstrate the facets that our model does well in but also accept that there will always be room for improvement in the future.

2. Related Work

There have been previous projects that have been done using deep learning methods to capture, and reproduce the human aspects of music. They have all fallen short in some aspect. Despite this there has been success and we hope to capture some of the elements that they did right in our own model, well also focusing on the elements most important to our exploration.

Eck and Schmidhuber explore the downfalls of Residual Neural Networks (RNNs) as a tool for music generation [2]. They state that RNNs lack the ability to store information relating to the past, so the network will lose track of where it is in a song. Drawing the problem back to the problem of vanishing gradients, Eck and Schimdhuber implement LSTM to a music generation model. LSTM successfully produced global structure and local structure, which provided influence into our final network decisions.

A research paper from Yue, Fu, and Liang investigates Res-RNN, which is a recurrent neural network with residual learning and shortcut connections [6]. The model attempts to solve the problem of vanishing gradients as well, but by referencing the hidden state. The findings show that Res-RNNs can produce as satisfying results as LSTM and GRU for music databases. Moreover, there is an improvement in training and predicting speed.

The most influential previously done studies on generating music using a Long Short-Term Memory (LSTM) neural networks. Skuli experimented with the idea in 2017, using LSTM layers, dropout layers, fully connected layers, and an activation layer [4]. To generate music, they submit the starting sequence to the network, then remove the first note and insert the output of the previous iteration. Subsequently, the model determines if the pattern is a chord or note and creates a Music 21 Stream object with list as the parameter. Following conversion from a stream to a MIDI file, the generated music can be played. One flaw in the model is that the network is not big enough to avoid false notes and imperfect melodies. In addition, the model only supports music with a single instrument. Despite these limitations, Skuli's model does provide hope for our model and it capturing the elements that we wish to emphasize.

The other studies on the topic of music generation through neural networks greatly influenced our model creation. We will create our model with those previously done in mind and take heart to the changes made. Our project experiments with a model containing LSTM layers as the main nodes using others in support just as those before us found out is best. We hope that by diverting from the other projects in our choice to use drum and rhythm pieces as input will allow us to focus on a specific set of music features and do them well.

3. Proposed Method

Our methodology will be naturally segmented into three parts, the pre-processing, training and the song generation. All of these three parts follow the general model example set by LSTM models that we observed in natural language processing and in some of the related music work that was done before us. By seeing each song as a sequence of vocabulary "words", we were able to use the general outline of these previous models while also adding some corrections for the musicality of it all.

3.1. Pre-Processing

In the pre-processing stage of the project, we broke down our input data set of 1800 drum and rhythm MIDI files into the input for our model. These Midi files were composed of various length files all played by drums and other percussion instruments. These files were specifically chosen due to there ability to be broken down into components that can be fed into a deep learning model.

To break these files into the components required the Music21 library was used. Using this library to parse each music file, we could break down each drum beat into our desired series of strings. Each string obtained from this breakdown represented a specific note, chord (A collection of notes) or rest that composed the beat overall input beat. From this string breakdown we created a mapping between the strings and there integer values so that they could be fed into network.

```
{0.0} <music21.note.Note C>
{0.25} <music21.note.Rest rest>
{1.5} <music21.note.Note C>
{1.75} <music21.note.Rest rest>
{2.0} <music21.note.Note C>
{2.25} <music21.note.Rest rest>
{3.0} <music21.note.Note E>
{3.25} <music21.note.Rest rest>
{4.0} <music21.note.Note C>
{4.25} <music21.note.Rest rest>
{5.5} <music21.note.Note C>
{5.75} <music21.note.Rest rest>
{6.0} <music21.note.Note C>
{6.25} <music21.note.Rest rest>
{7.0} <music21.note.Note E>
```

String representation of the note objects composing each music stream produced from the MIDI file

Using the vocabulary of notes provided to us from the parsing of the MIDI files, we would begin to prepare them for feeding into the model. As we planned on using a embedding layer to reduce the dimensionality of the what ended up being 1200 categories being fed into the model, all of the notes had to be turned into an integer to represent their values. This was done with a dictionary containing the string names of the notes that was a key to each integer, and a array that contained the reverse mapping. Maintaining the actual values of the notes was particularly important for the generation of the final song so that during the prediction phase we could rebuild the songs based on the model output.

One advantage that we found while creating the data to be fed into the model was that by the nature of the music we did not need to create a padding character, which is commonly required in text models. Because all of the beats are cyclical over a certain period, all of them can be extended to the same sequence size, meaning that we did not have to worry about the padding in our data. Moreover, each of the sequences were the same size, allowing for uncomplicated batching.

3.2. Deep Learning Model

In the model stage of the project, we broke down our desired outcomes and requirements of the aspired model first, then explored networks that matched the necessities. At the conception of this project, one of the obvious choices was a simple multi layer neural network, which would be capable of taking each note in and returning the most probable next note based on the given previous note. Although the neural network would be a sufficient option, it does disregard one of the indispensable aspects of music that we hoped to explore – rhythm. Without knowledge of the notes that came before the predictions of the next note, capturing the natural flow that music creates in its stream of notes would be difficult.

The challenge with rhythm meant that the model chosen required another condition; the model must be able to remember the previous notes and compile that information to the calculations of its own value. As a result of this requirement, a recurrent neural network was the clear solution. Taking advantage of the resources available to us and the projects that have done in a similar capacity previously, we decided to implement a LSTM network for our predictions.

The final model layout that we ended on was in three parts:

- An Embedding layer
- A LSTM Layer
- A Dropout Layer
- A LSTM Layer
- A Linear output layer

Based on the performance of some of the preliminary models that will be mentioned later, we went with a 5 layer model. The two LSTM layer model allowed us to have the flexibility to fit our data set, but the inclusion of a dropout layer in between allowed us to maintain a level of robustness in the predictions in the training process. Lastly, there is a Linear layer which provides us with our output.

3.3. Song Generation/Prediction

The final section stage of the process is the generation of the final song, this process happened in two steps. The prediction of a sequence of notes given the trained model, and then the transfer of these predictions to the resulting song.

To predict the final outcome we had to start with a starting note. As all of the note sequences that we fed into the model all started at note 0 we decided that to get the best results of the final training model we would use the most prominent starting notes to feed into our final song outputs. Feeding in this starting note as a sequence we then built of this one note adding the last note of the models output for each input as the sequence grew. To prevent repetition that we found quite prominent in our first songs, instead of taking the most probable note with torch max we opted to pick each subsequent note using the distribution from the softmax of the logits. This small step allowed the songs to stay true to the patterns that it had picked up during training, but also prevented the degeneration into a two note sequence.

The final step of the generation was to transfer our output integers from the model and into the series of notes that make up the predicted MIDI file. From our data processing step we had the integer to string table for the notes so

Hidden Layers	Validation Accuracy
1 layer with 128 nodes (lr = .001)	8.96%
2 layers with 32 nodes (lr = .001)	17.02%
2 layers with 32 nodes (lr = .0005)	38.05%

Table 1. Validation Accuracy

this process was fairly standard. As we did not assign any lengths to the notes in our vocab all notes, rests, and chords were assigned to be half-notes. Then taking this array of notes, chords and strings we turned them into the MIDI file outputs of the model.

4. Experiments

Based on previous works and our own research, we decided to implement a RNN with LSTM layers to generate our musical sequences. However, after this step, we still needed to experiment with different model parameters and hyperparameters like learning rate, number of hidden layers, and number of nodes within each hidden layer. After we decided on our model, we also needed to experiment with ways to actually create some sort of musical output.

4.1. Model Parameter Experimentation

Our first step was to experiment with various model parameters to see which combination would give us the best possible accuracy on our validation set. Originally, our goal was to have a model that could accurately predict the next note of a sequence given a sequence of notes as an input. Initially, we started with a 1 hidden layer RNN (with LSTM) and 128 nodes within this hidden layer. The validation accuracy on this combination of parameters was very low, around 8.96%. Obviously this was not ideal, so we decided to make a variety of changes to the model. We added another hidden layer, with each hidden layer having 32 nodes. Additionally, we added 90% dropout to prevent over-fitting. With these additional model parameter changes, the validation accuracy increased to around 17%. In both models, we implemented an ADAM optimizer with a learning rate of 0.001 with cross entropy loss. We then decided to experiment with the learning rate. By decreasing the learning rate from .001 to .0005, the accuracy of our model went up to 38%.

4.2. Model Output Experimentation

Despite testing out various different model parameters, including number of hidden layers, dropout rate, and the number of nodes within the hidden layer, we were running into similar problems for each model. Specifically, the sequence of notes outputted from the RNN consisted of a repeated sequence of only two or three notes. Obviously, the output was not ideal as we wanted to produce music with a coherent beat or pattern and we were unable to accomplish that with a regular RNN (with LSTM layers) and a softmax output function. Therefore the crucial question developed, how can we generate music with some sort of pattern?

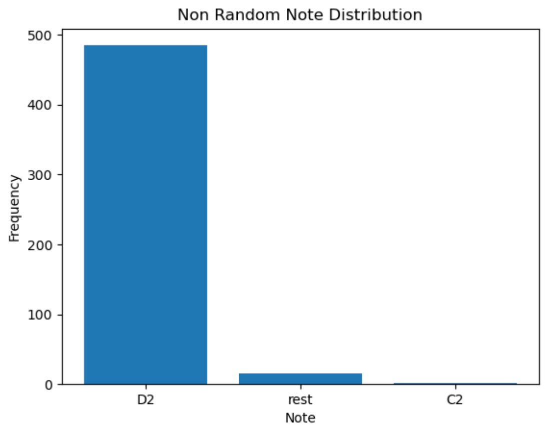

Figure 1. All the notes in a 500 note song generated without randomness

The main way we worked around the issue of repeating notes was experimenting with our output function. The typical approach would be to just select the note with the highest probability of being selected next (based on the probits from the softmax layer). However, as mentioned above, this approach was causing the same notes to be picked over and over again. Therefore, we decided to add some randomness to our output by sampling from a multinomial distribution. However, these predictions were not completely random because we sampled from the probability distribution of the probabilities generated from the softmax output layer of the model. This approach finally led to more variation in the types of notes that were outputted by the model. For example, a 500 note song generated without sampling from a multinomial distribution consisted of only three separate notes (Figure 1). With the same starting note, we were able to generate a song with 105 different notes using the multinomial sampling procedure (Figure 2).

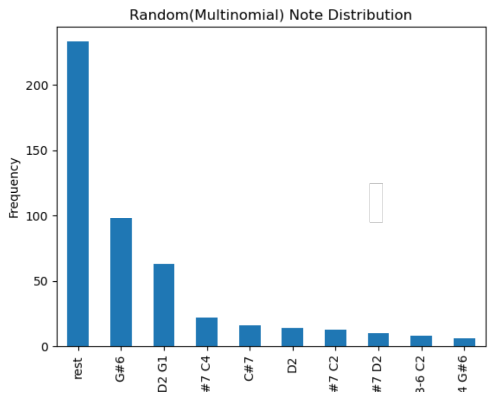

Figure 2. The top 10 (out of 105) most frequent notes in a 500 note song generated through a multinomial distribution

Additionally, we experimented even more with these randomized outputs by adding a measure called temperature into our randomized outputs (Figure 3). Temperature is a measure that controls the randomness of the prediction, with a higher temperature leading to more randomness. Formally, we use temperature to scale the logits (by dividing the logits by the temperature) before generating the probits from our softmax function. The equation for generating these probits is $q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$, where q_i is an outputted probability, z_i is a logit, and T is the temperature. We then sample from a multinomial distribution using these probits, similar to the method in the paragraph above. If we would want to replicate the method in the paragraph above, we simply set the temperature to 1. When setting the temperature to five, we are able to generate a 500 note song with 406 distinct notes with no note being used more than four times. A 500 note song generated with the same starting note without temperature produced 105 distinct notes and a 500 note song generated with the same starting note without randomness or temperature produced 3 distinct notes. Refer to figures to 1,2, and 3 to see a rough breakdown of the notes within songs for each output method.

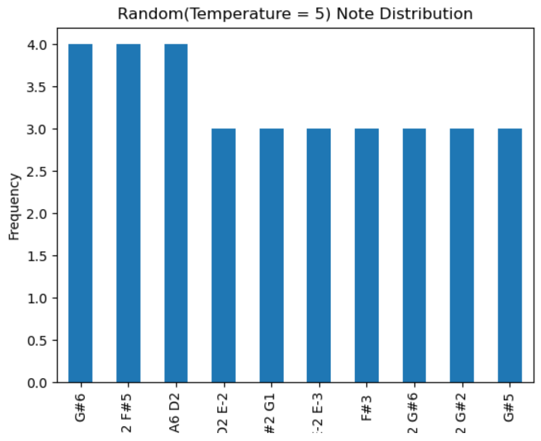

Figure 3. The top 10 (out of 406) most frequent notes in a 500 note song generated through a multinomial distribution with a temperature of five

4.3. Data Set

We looked into a variety different formats for music data. The decision to go with MIDI was due to the number of resources available to work with these types of files and the ease to convert these files into something that can be used as inputs into the deep learning model. The data set was obtained from a publicly available zip file that had be collected by the reddit user "Midiman", who used web scraping to collect them from multiple online sources[3]. Despite this set of midi files containing many different types of music we decided to focus on a subset that only contained drum, and percussion beats. This decision was meant to reduce the complexity of features being fed into the deep learning model. The data was composed of MIDI files classified into eight different genres: blues, country, indie, jazz, pop, psychedelic rock, rock, and soul. In terms of our train,test, and validation split, we allocated 75% of the data for training, 10% for validation, and 15% for validation. After finding our best model, we used all available data to generate our songs.

4.4. Software and Hardware

We used python scripts for data pre-processing, model building, model predictions, and processing outputs back into MIDI files. After we had obtained the model that we thought would do the best, we trained it overnight on one of our desktop computers on 50 epochs, which took around two hours. Specifically, the PyTorch and Music21 libraries were employed. Music21 was vital for the experiment because the library can parse the digital MIDI files and return the notes and the instruments that were used to play these notes. Essentially, the library is a helpful aid for musical analysis.

5. Results and Discussion

Here we will discuss the results of the model that we created as well as all of the discussion.

5.1. Parameter Selection

Throughout our experimentation, we reached varying levels of accuracy with different model parameters. Our highest accuracy achieved on the validation set involved a two hidden layer RNN with LSTM where each hidden layer had 32 nodes. We used an ADAM optimizer with a learning rate of 0.0005 and a cross entropy loss. One explanation for an increase an accuracy when decreasing the learning rate is that at higher learning rates, the model may have been skipping over some sort of global minimum.

Before decreasing the learning rate, we also increased our validation accuracy by going from a one hidden layer RNN (with LSTM) with 128 nodes to a 2 hidden layer RNN (with LSTM) with 32 nodes each. This extra layer increased the flexibility of our model allowing it to find more complex features within our sound data. However, with the added flexibility comes the possibility of over-fitting. To avoid this we added ninety percent dropout within the hidden layers so that the model does not rely on any specific neurons when training, which did decrease the signs of overfitting on our model.

5.2. Accuracy And Model Assessment

The assessment of our model was broken down into 2 stages. The nature of music means that there is not always a correct next note to output, instead there are many different notes that can follow and sound good. This caused issues in our initial accuracy assessment that we will address below, and additionally we will discuss how this changed our assessment of what was a good model.

In the beginning assessments of our model we used an accuracy measurement that was very dependent of perfect predictions. This resulted in accuracy measurements that followed a strange trend in our training and validation accuracies. Many times, our maximum training and validation accuracy was in one of the first five epochs and there was a steady decrease in accuracy for the following epochs. This decrease in accuracy contrasted our epoch to epoch decrease in loss as seen in figure 4. From this our ”best” model using this assessment strategy our best model achieved an accuracy of 40% on the testing set by using a strategy of predicting rests, which is the most occurring note, continuously.

This perfectly accurate model might not actually lead to the most coherent sounding music being generated by the model. As discussed above by using accuracy as a model measurement the predictions were highly repetitive and quickly degenerated into a 2 note pattern. This did demonstrate that our model was picking up on the rhythm of the music, but also unfortunately prevented any creativity in the music created. This issue will be further explored in the next segment.

To correct for this we decided to move from this measurement of accuracy and instead focus on decreasing the loss of our validation set as the measurement for the best model. This meant that as long as the most likely notes were being predicted next we would continuously improve our model. This made sense to move to because this allowed leniency in the predictions where other notes make as much sense as others. So despite not predicting the correct note as much, the resulting songs of this new method of assessment sounded much better in the finished product.

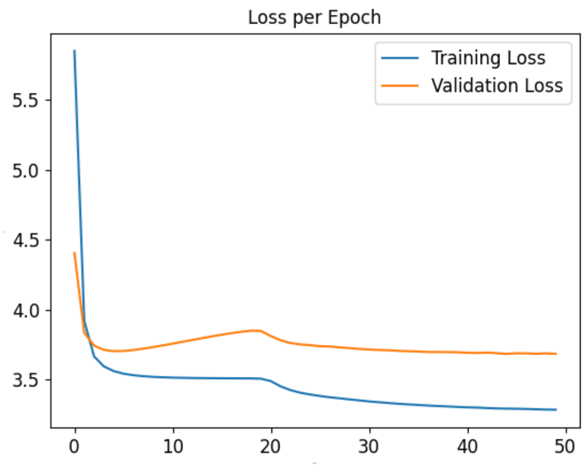

Figure 4. Training and Validation loss for final model over 50 epochs

In the figure above you can see the final loss descent and a also a case of double drop descent when we trained our final model on 50 epochs.

5.3. Song Prediction Results

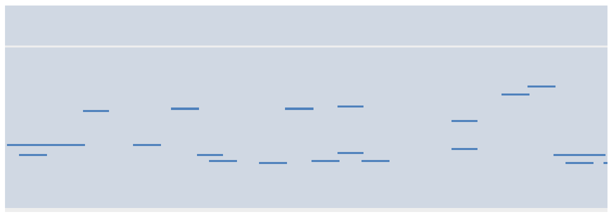

Figure 5. Image of a generated song segment and its pieces

As mentioned in the experiments section, a regular softmax output layer resulted in the model creating sequence with little variation in notes. In fact, when attempting to generate a song with a length of 500 notes, the song only has three distinct notes, with a majority of the notes belong to only two classes. In order to produce more realistic sounding music, some randomness was needed. When we sampled our outputs from a multinomial distribution using probabilities generated from the model, we were able to generate songs with much more variation in notes. This allowed our model to generate interesting patterns in notes without repeatedly using the same notes. One explanation for the lack of variation in the original sequence output is the occurrence of extremely common notes in the training data. For example, the rest note is one of the most common notes in the training data, and the model might be defaulting towards these notes. Many times, the softmax output function assigns the highest probability to these common notes. Adding randomness allows us to break any sort of cycle of common notes while still respecting the probabilities returned by the model since our sampling procedure utilizes these probabilities. We also looked into increasing our randomness through temperature measure, which scaled our logits before calculating our probits. However, we decided to go with a temperature of one, essentially equal to not scaling logits, because we did not want to introduce too much randomness into our generated sequences, just enough so that the same notes were not being repeated over and over again. Having too much randomness would cause our generated sequence to stray too far from the actual probits generated from our model outputs. We still want the notes outputted to be based mostly on the model, with randomness only being used to break repetitive cycles.

6. Conclusions

In conclusion, we were thrilled to create a model that generated actual music with a beat and pattern. There were many issues we ran into, including low validation accuracy and very repetitive music. We were able to come up ways to alleviate these problems by adjusting different hyperparameters and sampling our predictions from a multinomial distribution. It was a great experience to apply some of the concepts we learned in class to work with RNNs (with LSTMs) and audio data. The data pre-processing step was very challenging, but we all ended up learning a lot about a possible ways to work with audio data in the future. There are definitely many avenues for improvement.In the future, it might be possible to have the model generate music based on a specific genre or request. Additionally, we could try out different formats of audio data and see if they produce better or worse results than the MIDI encodings we used in our current project.

Our findings are comparable to the conclusions from related work. We encountered some of the same weaknesses of previous models. However, with experimentation, we were able to address these weaknesses. The limitations of RNN, including not remembering previous notes, were confronted by adding a LSTM hidden layer.

7. Acknowledgements

As our group had no prior deep learning experience coming into the class, we would like to acknowledge Professor Raschka for his indispensable contributions to our learning. Furthermore, we would like to acknowledge the originator of the dataset, the infamous ”MIDI MAN” who made it one of his life goals to accumulate these MIDI files in a easily downloadable zip file. Without the resources Professor Raschka and the originator provided, we would not have been able to conduct our experiments.

8. Contributions

8.1. Prithvi Tanna

Prithvi created the original model, that was eventually formed through his own work in tuning the layers, into the final model. He created the accuracy metric for the training and also created the temperature function that was used as the solution to the repetition of highly probable notes. Prithvi rote large chunks of the paper and also was apart of brainstorming the audio formatting issue and how we would get the digital files into a format that would be usable input into the model. Additionally, Prithvi spent large amounts of time working with Cannon to fine tune the final model in terms of hyper parameters.

8.2. Nicole Athanasiou

Nicole put enormous amounts of work into the presentation of the final paper and presentation, did the research into all of the previous projects that have touched on this issue and looked into what we could do better. She made major code contributions include writing the base song generation by using the tuned model. Then she took that array of strings and turning it into a playable MIDI file in the predict.py file using the Music21 library. Additionally, she provided a large amount of the overall structure and deadlines of the assignment.

8.3. Cannon Lock

Cannon did a large amount of the research into data sources and what characteristics these sources had that made them feasible input into the model that we were going to create. Cannon wrote the majority of the MIDI parsing code and worked with Prithvi to create the training code and the final model. He looked into a lot of the similar projects and also helped research fixes to the shortcomings of our model as they came into light, well working with the others to implement these changes.

9. Code

The code for this project can be accessed at the following GitHub repository: https://github.com/CannonLock/PyCode.

References

[1] J. Brownlee. How to evaluate generative adversarial networks. 2019.

[2] D. Eck and J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. 2002.

[3] Midiman. The drum percussion midi archive, 2016.

[4] S. Skuli. How to generate music using a lstm neural network in keras. 2017.

[5] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova. Music transcription modelling and composition using deep learning, 2016.

[6] B. Yu, J. Fu, and L. Jun. Residual recurrent neural networks for learning sequential respresentations. 2018.