Aaron Luo
A13782379
Tu
Cogs181

Utilizing Neural Nets in Music Generation

Abstract

Music like most art forms is generally considered to be difficult to produce computationally, as it is hard to determine what combinations of sounds is considered to be good music or even music at all. However, recent advances in machine learning have found ways to quantify complex music theory concepts, which allows algorithmic generation of music that would sufficiently imitates music written by humans.

Introduction

Neural networks have been used to allow for the computational production of artistic works that were previously considered to be purely within the human creative domain such as painting, poetry, and creative writing. Music, however has proven to be somewhat more challenging, as while superficially musical works can be similar to other forms of character generation such as name generation, chat bots, and writing imitation, musical generation necessitates the quantization of factors beyond simply music notes vectorization. This is because timing is important in terms of the quality of music and therefore factors such as note duration and onset time are important to consider. Other factors such as volume, otherwise known as dynamics, as well as chords also have significant contribution to musical complexity and therefore should be integrated in some form if any subsequent music generation is to be considered totally successful. Furthermore, imitation of live human performances presents additional problems as what often makes music sound good is the performer's personal interpretation of what is objectively written in the music, something that can be difficult for a computer to capture. This project will attempt to tackle some of these problems, namely the note duration, onset time and volume. Generally with natural language processing problems, recurrent neural networks (RNN),as well as its other variations such as long-short term memory (LSTM) and gated-recurrent units (GRU), are used, as RNN's are designed to recognize patterns in sequential/temporal data in which every data point is assumed to be dependant on the data points that came before it. Furthermore, RNN's have a distinct advantage against other neural networks such as convolutional or feedforward neural nets in that it can not only store the memory of previous data but can also be used on datasets where the size is initially unknown during preprocessing, making it useful for more dynamic forms of sequence prediction such as real time/online. However vanilla RNN's are disadvantaged in that they have a tendency to drop important information if the sequence is too long due to the vanishing gradient problem in which the gradient may be reduced into nothing as it back propagates through previous data. GRU and LSTM solve this problem by regulating data and throwing out information that is regarded as extraneous. These traits should be useful when processing musical data which, in its essence, is similar to character data used in natural language processing, even with its added complexity.

Methods

In this project we will be training our model using Beethoven's sonatas in an attempt to generate music that would be similar to the style of music he would produce. All 32 of Beethoven's sonatas were

gathered on 32 seperate midi files, totaling to 8 hours, 54 minutes, and 17 seconds of music. The use of this particular file type is important because it allows us to gather important data regarding the music without using any sort of spectroscopy, which is a whole machine learning problem in of itself. Said data would be gathered using the tools provided by the music21 Python library.

Data will be trained using random sequencing through a many-to-many model of two different types, namely vanilla RNN and GRU RNN. Vanilla RNN was chosen to provide a baseline performance model to evaluate other RNN types, and should theoretically have a lower performance rate than its other versions. GRU as a more specific form of LSTM was chosen due to it being a RNN option with a theoretically higher performance, as it prevents the disappearing gradient problem due to its integration of forget gates while at the same time being more computationally than regular LSTM models due to a reduced complexity. Both models are to be implemented in the same way, i.e., with a batch size of one, a hidden layer of with 100 neurons, and a linear transformation output layer. LSTM was initially going to be tested due to its theoretical effectiveness especially for music datasets as well as to supplement the performance of the GRU model, but due to time constraints was unable to properly implemented and thus was left it out of this project.

In addition to RNN model type, the sequence length and learning rate hyperparameters are to be manually tuned over three different values; more specifically, [32,100,200] for sequence length and [0.001, 0.005, 0.05] for learning rate. Learning rate was chosen due to its easy implementation as well as its general significance in training rate and accuracy. Higher learning rates should cut down on processing time at the cost of accuracy. Sequence length was chosen due to the sequential nature of the neural net models used for this project, and therefore tuning sequence length should have a noticeable effect even though the general significance of sequence length is somewhat ambiguous. Conceptually, increasing sequence size should return a higher accuracy rate at the cost of processing time, as larger sequences allow for more predictors while at the same time taking longer to train.

The dataset used was compiled specifically for this project based on the information extracted from the midi data files. The data file contains 3,571,277 data points partitioned into 32 sections based on their respective songs. Each partition is to be processed separately, as notes from one song should not be a good predictor for notes in a different song. The data was then split into four categories, namely, notes, duration, onset time, and volume. Each of these categories are of the same length, as they each represent different parts of the same song, but differ in number of features/complexity. To elaborate, the notes dataset, which refers to the pitch key of note in the song (e.g., C4, F#5, D-3), has 95 labels, the duration dataset, which refers to how long each note is played, has 79 labels, the onset time dataset, which refers to the time the note is played, has 22,422 labels, and the volume dataset, which refers to the loudness of the note, has 109 labels.

Experiment

All of the music contained within the dataset was normalized by transposing them into the same major or minor key, namely C Major or a minor. These particular keys were chosen because both C major and A minor have the same key signature of no sharps or flats, which in of itself should already reduce some complexity. Furthermore, all rests and chords were removed during preprocessing in order to reduce feature complexity that would come with including them. For reference, the data set that takes chords into account ended up with approximately 1382 features, while using only notes resulted in 95 features. This is because chords are essentially any given combination of notes but are treated as individual entities when

used in a dataset, therefore introducing a large amount of unique data points. While chords, as mentioned above, are important contributors to musical complexities in any given song, keeping them increases processing time by a significant amount, as well as, based on my research in other similar projects, could potentially complicate the training process and result in music that sounds somewhat incoherent. Therefore, for the purposes of this project, melody generation/replication was the primary focus, and additional harmony generation will be left to future endeavors.

Notes were extracted from the midi files and put into datasets in the order they were extracted, after which they were processed to determine the number of unique characters as well as to create a one-hot encoded feature vector from the note dataset created earlier. Duration time was processed in a similar way despite superficially appearing to be a numerical data type, as each data point has a fixed duration determined by each note's type (quarter note, eighth note, half note), and is therefore effectively categorical. Onset time was likewise treated as categorical for a similar reason in that the notes can only be played on certain beats and cannot occur on any arbitrary time point. Volume was regarded as categorically as it was contained within a certain range, and therefore processing it as such was more straightforward processing it as if it were numerical.

Hyperparameters were individually tuned and optimized to the notes datasets, with training loss, training loss graphs, and processing time collected for each. The hyperparameters were not optimized to the other three datasets due to time constraints, so therefore the optimal parameters for the notes dataset was assumed to be representative of the optimal parameters for all the datasets. This is something that should be fine with the duration and volume datasets, as they have a similar amount of features as well as the same length for obvious reasons, but may not work as well with the offset dataset, which has a significantly higher level of complexity.

Results

GRU loss	seq_len = 32	seq_len = 100	seq_len= 200	GRU Time	seq_len = 32	seq_len = 100	seq_len= 200
lr = 0.001	2.349	2.290998	1.1035	lr = 0.001	334.8235	248.506088	2140.3199
lr = 0.005	0.95305	0.25664	0.17979	lr = 0.005	256.99	847.037	1421.23
lr = 0.05	4.551	4.089013	3.9003	lr = 0.05	340.268	227.353712	2106.1316
RNN loss	seq_len = 32	seq_len = 100	seq_len= 200	**RNN Time**	seq_len = 32	seq_len = 100	seq_len= 200
lr = 0.001	2.4805	1.6371	1.1257	lr = 0.001	308.145	1147.6309	2536.0425
lr = 0.005	0.28271	0.25584	0.151266	lr = 0.005	879.692	804.961	2181.407
lr = 0.05	5.62988	4.2951	3.817	lr = 0.05	313.85579	1111.90182	2546.68

As seen with the chart above, increasing sequence size returned higher training accuracies at the cost of significantly higher training time, a result that more or less aligns with what was predicted. It is notable that across all three times and learning rates, the training loss stayed relatively similar, but at the same time consistently introduced a large spike in training time, especially in between a sequence length of 100 and a sequence length of 200. Interestingly, the lowest sequence length actually returned a slightly higher loss rate as well as a slightly longer training time in comparison to the next sequence size of 100 when the learning rate was not 0.005, suggesting the existence of an optimal learning rate. This is supported by the fact that a learning rate of 0.005 had the best performance across all sequence lengths by rather significant margins. This is surprising as intuitively higher learning rates should result in a faster training time at the cost of accuracy, and that optimization came in the form of finding the most efficient tradeoff between accuracy and processing time. Perhaps what is even more surprising is how close vanilla

RNN and GRU were in terms of training accuracy. Vanilla RNN was even more accurate than GRU when the learning rate is 0.005, but did perform slightly worse outside of that learning rate. Where GRU was clearly more advantageous in was in terms of training time, as vanilla RNN consistently took significantly longer to return what turns out to be somewhat similar results as GRU, therefore demonstrating an overall higher performance rate when the GRU model is used as opposed to the vanilla RNN model, aligning with what was predicted.

Conclusion

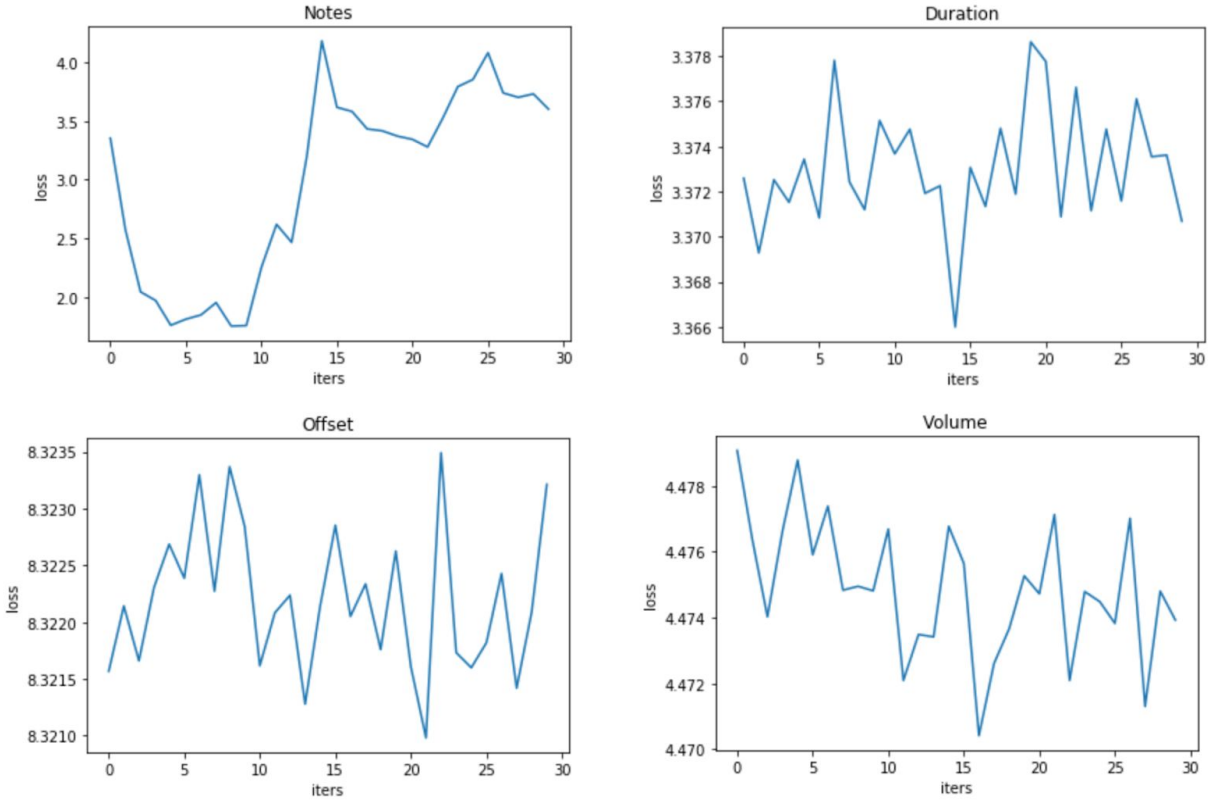
Although the results returned by hyperparameter optimization look pretty definitive, they, unfortunately, did not translate well when used to train all four datasets at the same time. Below is the alpha numerical transcription of the final result using the hyperparameters that returned the best accuracy rate during optimization, that is, a GRU RNN model, a sequence length of 200, and a learning rate of 0.005. The datasets are in order of notes, duration, onset time, and volume.

```
iter:2999/3000 loss:3.602207593917847
generated sequence: ['E6', 'A6', 'E6', 'A2', 'D6', 'F3', 'F6', 'G6', 'D6', 'G#4', 'F6', 'B2', 'E3', 'E6', 'D6', 'E6', 'D
6', 'E6', 'A6', 'G6', 'F6', 'E6', 'E-4', 'E6', 'D6', 'E-4', 'C6', 'E3', 'G#4', 'A6', 'G#4', 'B5', 'B2', 'D6', 'E2', 'F#
4', 'D3', 'D6', 'E6', 'A4', 'E4', 'C3', 'F#4', 'E3', 'C4', 'D4', 'E-6', 'B3', 'F#3', 'B2', 'E3', 'F#4', 'C6', 'G#4', 'E
3', 'B2', 'E3', 'D3', 'D6', 'G5', 'D3', 'D6', 'E-4', 'F#2', 'D3', 'E6', 'A2', 'E4', 'E-4', 'E-3', 'F#3', 'B2', 'E-3', 'A
4', 'E4', 'E3', 'C#5', 'B4', 'G#4', 'B4', 'B5', 'G#4', 'F#4', 'E-4', 'C#5', 'B4', 'G#4', 'G#4', 'C5', 'C6', 'F6', 'D6',
'B4', 'B2', 'D3', 'F6', 'E6', 'C4', 'G#4', 'E3', 'E-4']

iter:2999/3000 loss:3.370688729286194
generated sequence: [Fraction(13, 3), Fraction(5, 3), 1.75, Fraction(2, 3), 1.75, Fraction(1, 3), 16.0, 8.0, Fraction(13,
3), 1.0, Fraction(8, 3), 2.75, 2.5, 4.0, 2.75, 2.0, 3.0, 0.25, 2.0, 2.25, 1.25, Fraction(23, 3), 4.0, 1.5, 12.0, 3.25, Fr
action(2, 3), 0.25, 2.25, 3.25, 4.0, Fraction(2, 3), Fraction(23, 3), 1.75, Fraction(1, 3), Fraction(2, 3), 8.0, 5.5, 4.
0, 4.0, Fraction(8, 3), 2.25, 8.0, Fraction(1, 3), Fraction(23, 3), Fraction(1, 3), Fraction(5, 3), Fraction(13, 3), 1.7
5, 3.0, 8.0, 3.0, 2.25, 0.5, 0.75, 1.75, 2.75, 5.5, Fraction(13, 3), 5.0, Fraction(23, 3), 2.5, Fraction(23, 3), 1.25, 0.
5, Fraction(7, 3), 2.0, 3.25, Fraction(5, 3), Fraction(23, 3), 3.25, 16.0, Fraction(7, 3), Fraction(11, 3), 3.0, 3.25, 3.
0, Fraction(8, 3), 3.0, 0.5, 1.75, 16.0, 4.0, 2.25, 0.75, 2.75, Fraction(1, 3), 12.0, Fraction(2, 3), 0.25, Fraction(5,
3), 3.0, 2.0, 6.0, Fraction(8, 3), 2.0, 5.5, Fraction(7, 3), 0.25, 1.0, Fraction(1, 3)]

iter:2999/3000 loss:8.32321307182312
generated sequence: [657.25, 951.0, Fraction(6064, 3), 316.5, 2058.0, Fraction(6113, 3), 1761.0, 1937.0, Fraction(5413,
3), 408.0, Fraction(5864, 3), 1950.0, 900.0, 1979.0, 513.0, 174.0, 1809.0, 170.0, Fraction(2605, 3), Fraction(6332, 3), 1
687.0, Fraction(5567, 3), 795.75, 332.5, Fraction(2545, 3), 1088.5, 150.0, 2148.0, 2136.0, 828.0, 144.0, Fraction(2849,
3), 1413.0, Fraction(605, 3), 932.75, 329.5, Fraction(5753, 3), Fraction(6124, 3), 283.0, 1340.5, 959.5, 1740.5, Fraction
(6127, 3), 1625.0, 1106.5, 1192.5, Fraction(6155, 3), 1269.5, 782.5, Fraction(6442, 3), Fraction(6406, 3), 891.5, 304.0,
287.5, 80.5, 1124.75, 181.75, Fraction(5605, 3), Fraction(4366, 3), 898.5, 286.5, Fraction(5855, 3), Fraction(6434, 3), 2
58.0, 342.5, 277.0, 683.25, Fraction(5771, 3), 1682.0, Fraction(4571, 3), 515.0, 1173.0, 1854.0, 67.0, 295.5, 2104.0, 175
1.0, 1795.0, 82.0, 1205.0, 1189.0, 1996.0, Fraction(4684, 3), 935.5, Fraction(5731, 3), Fraction(5779, 3), 134.0, Fractio
n(2491, 3), 1795.0, 963.0, Fraction(6118, 3), 495.0, Fraction(4535, 3), 1583.0, 1820.0, 660.0, 753.75, 1578.0, Fraction(4
250, 3), Fraction(4817, 3), Fraction(6253, 3)]

iter:2999/3000 loss:4.473929328918457
generated sequence: [49, 67, 79, 41, 27, 28, 79, 26, 33, 93, 89, 109, 32, 52, 104, 99, 110, 79, 106, 28, 26, 67, 45, 81,
41, 81, 69, 22, 26, 61, 103, 69, 71, 42, 40, 96, 88, 76, 105, 92, 59, 44, 83, 46, 60, 66, 39, 27, 90, 56, 52, 57, 104, 6
1, 66, 57, 78, 59, 75, 39, 109, 44, 64, 108, 40, 104, 75, 57, 39, 58, 67, 29, 41, 76, 45, 70, 92, 44, 26, 104, 28, 26, 3
5, 102, 55, 98, 64, 61, 74, 22, 80, 65, 50, 41, 28, 42, 110, 83, 48, 57, 81]

Processing time: 5937.551033973694
```

While it is understandable why the onset time dataset did not seem compatible with the hyperparameters optimized using the notes dataset due to it having significantly more features than it, it is rather surprising that the duration, the volume, and even the notes dataset itself still turned in relatively high error rates despite all having very similar dimensions.

A possible explanation for this is that the neural net was adjusting its weights to accommodate for the loss associated with the processing of the data sets other than the notes dataset. Below are the loss

graphs for each data set.



From this, we can see that the notes dataset loss begins to trend downwards like a typical loss curve but begins to curve back up at around 100 iterations. The loss curves of other datasets have no trends, which most likely suggests that there was something with the way the program was coded, as although each dataset was meant to be used to train seperate neural nets, it would seem that they were affected by each other in some way. Another possible explanation is that the simple neural net used over the course of this project was insufficient to make proper predictions on the given datasets. Further experimentation would need to be done to sufficiently figure out the problem.

Unfortunately due to time constraints I was only able to finish a fraction of what I originally wanted to do with this project, such as finding more efficient ways to work with chords beyond what I have found as well as learning how to implement LSTM, something I had much trouble figuring out due to my unfamiliarity with the PyTorch and TensorFlow libraries. Therefore, if I were to continue this project in the future, this is what I would primarily focus on.

References

Sharma, G., & Sharma, G. (2018, October 17). Music Generation Using Deep Learning. Retrieved from https://medium.com/datadriveninvestor/music-generation-using-deep-learning-85010fb982e2

Sigur. (2017, December 07). How to Generate Music using a LSTM Neural Network in Keras. Retrieved from https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

Lswari, D. (n.d.). Description of Hyperparameters for Recurrent Neural Networks. Retrieved from https://gist.github.com/DavidIsrawi/6c45744c12a4f8fc08bd5b8f7f9e06d8
Böhm, T., & Böhm, T. (2018, August 15). How to optimize Hyperparameters of Machine Learning Models. Retrieved from https://towardsdatascience.com/how-to-optimize-hyperparameters-of-machine-learning-models-98baec703593
Music21 Documentation. (n.d.). Retrieved from https://web.mit.edu/music21/doc/index.html
Neural Net code is based on the code assigned in HW 6