NeuralDrummer

A neural network for generating drum tracks for songs.

Abstract

The drum kit is an essential instrument in modern popular music. Drums often provide the rhythmic foundation of the music, often interacting with other instruments to strengthen the rhythmic structure. In this report, I explore whether a computer program can emulate the playing of a drum kit, and whether such behaviour would be considered creative. I model key characteristics and metrics of music in an attempt to model the intricacies that inform how a real drummer plays and reacts to music. I present a system that can generate an appropriate drum track from any given input using metrics extracted from the music.

Introduction

It has already been shown that AIs (Artificial Intelligence), specifically neural networks, are capable of producing music as shown in MuseNet (Pal, Saha, and Anita 2020) to at least some level of competency. Is it possible that by limiting the generative content to just drums, an AI would be able to create drum tracks on the same level as a human player? And by what process would the tracks be created?

Drums and percussions instruments do not have pitches, which eliminates a part of the challenge that exists for generating music with tonal instruments. It is easy for humans to detect dissonant arrangements but teaching a generative system to avoid such issues is much harder. You could attempt to hard code the system to not generate dissonant sounds, but this can limit the creativity of such systems.

When a human listens to a random arrangement of note lengths and pitches presented as a piece of music, it is easy for them to know that it *sounds wrong*. Is it possible to teach a generative system that certain arrangements sound wrong without it being about to hear and form an opinion on what it is generating? Some sort of fitness function could be used, but such function would be ‘calibrated’ to what the programmer found sounded good and may not be representative of the views of others.

Background

There does not seem to be a system that exists that does the same thing as the system I have created. An existing neural network designed specifically for drums is DrumsRNN (Magenta 2016), which can be used to generate a short loop that continues a starting drum pattern input. It uses a timestep system, where each drum note’s start time is quantised to the nearest beat. It does not take into account any MIDI information at all, and the input to the system consists of only previous drum timesteps.

A more generalised system that can generate music is Pyzzicato (Petit 2021) which generates on a per-note basis (not timesteps). This approach is more suited to generating music with more human qualities, as timesteps lead to all notes starting on beats and feeling very robotic. This per-note generation would work well with generating instrument pieces, where notes have a start *and* end, but percussion instruments whose notes only have a start makes this approach less relevant.

There are systems that generate music that do not rely on neural networks or deep learning, but I believe that the output of such systems is limited in both the creative aspect, and its ability to generate an original drum track that react to the rest of the music. An example of this is Dunc's Algomusic (Marasmusine 2016), which is capable of generating songs with multiple instruments and sections given a random seed. The outputs of Algomusic can be seen to have a high novelty, but the lack of knowledge as to what a note or pitch, etc., actually is hinders the perceived quality of the output.

System like MuseNet (Pal, Saha, and Anita 2020) and DrumsRNN use deep learning neural networks to generate outputs, specifically, they use an RNN (recurrent neural network) architecture to generate outputs given the previous inputs. Even more specifically, both MuseNet and DrumsRNN both use LSTM (long short-term memory) units within their architecture. LSTM units have the advantage of remembering long-term dependencies, which is especially useful when it comes to generating music and remembering which notes have been played previously.

The process for generating a drum track can be similar to generating instrument tracks, but with several key differences:

- Most instruments have a large range of notes that can be played, which may not be suitable for one-hot encoding. Percussion instruments have a relatively low number of unique notes (i.e., sounds), making them much more suitable for one-hot encoding.
- With certain instruments, only one pitch can be played at one time. Percussion instruments, such as a drumkit, can play multiple sounds at once.
- Certain ways of playing instruments, such as arpeggios and trills, are not applicable to percussion instruments and do not need to be modelled.

Using these differences, we can create a specialised system that excels in creating drum/percussion tracks. By using discrete timesteps, we can think of the problem as a time series problem. Much like predicting the next daily temperature using previous days, we can predict the next timestep using previous timesteps. A disadvantage of this is that outputs can only ever be on a beat/timestep, meaning that there is some limitation to the intricacies of the resulting output, but this seems like an acceptable limitation given the scope of the project.

Methodology and Design

The project was implemented using Python, specifically using TensorFlow/Keras (Google 2019) to train the neural network, and the Mido (Mido Contributors 2017) and PrettyMidi (Raffel and Ellis 2014) libraries to handle interactions with MIDI files and data. To train the neural network, I used the Lakh MIDI Dataset as collated by Raffel (2016), specifically I used the “Clean MIDI subset” version. I used this version as it contains matched MIDI files with their title and artist and the contained files are more likely to be of better transcription quality.

To begin, we pre-process all MIDI files to token strings that can be easily read and saved to the disk. This step is not necessary for the network to run (you could instead process the MIDI files as they are requested when training the neural network within the “DrumSetSequence” class) but it saved a lot of time to pre-process the files and save the resulting tokens to disk instead of having to pre-process the files and store them in memory each time.

MIDI files have their drum tracks extracted and each note's start time is quantised to the nearest eighth note in a similar fashion to DrumsRNN (Magenta 2016). This allows the neural network to generate notes in sequence using a fixed timestep. The following is an example of what a token string generated from a MIDI file might look like:

```
42,36,i0.0,-1,42,i0.6,-1,42,38,i0.6,-1,38,i2.2,-1,42,36,49,i-0.5,-1,42,i0.6,-1,42,38,...
```

In this example, we can see that each MIDI note is encoded as a number representing its MIDI pitch value, which for drums, is mapped to a specific drum/percussion sound. These pitches can be looked up using the General MIDI 1 specifications (MIDI Association n.d.). Each pitch/token is separated by a comma. There are two special tokens that get read into the program differently. These are the tokens that start with an "i" or are "-1". The "-1" token tells the parser that the array has moved on to the next timestep. In the example, "42", "36", and "i0.0" occur on the same timestep, with "42" and "i0.6" occurring on the next timestep.

By abstracting the input MIDI into metrics, we should be able to capture the essence of the track without losing too much information. This is achieved via "intensity" tokens. The "i" tokens are used to save an intensity for each timestep. Intensity is defined for each timestep as how many MIDI notes start on that timestep. Intensity accounts for all MIDI notes from all instruments included in the file, not just drums. The intensity value is rounded to one decimal place to help save space in the generated file. For each MIDI file, intensity for each timestep is initially computed as the sum of all notes that start within a timestep. Then, outliers (if any) are removed and replaced with the average of the previous and next valid intensities.

The intensities are then normalised by subtracting the mean and dividing by the standard deviation for each intensity. Finally, the intensities are offset by –1, meaning that the intensity for a timestep is actually the intensity that occurs in the previous timestep. This allows the neural network to generate the next timestep given its intensity, rather than only having the intensities before the timestep it is asked to predict. Allowing the neural network to make use of the intensities should allow it to account for changes in intensity of other instruments when generating the next timestep.

This processing is performed once for each MIDI file in the input dataset, and the combined results are saved to a file on disk.

After the pre-processing steps described above, we can use the token arrays to train the network by windowing the input data. Windowing allows us to take a consecutive series of N timesteps from a random point in any of the input token arrays. We use the first N-1 timesteps as input to the network, and the final timestep as the target output. By repeating this process, we can train the network to generate predictions for the next timestep given N-1 previous timesteps.

The neural network's architecture consists of multiple layers of LSTM units, followed by some Dense unit layers to convert the LSTM's output into one-hot encoding format (same as the inputs apart from intensity). The network requires initial starting inputs of size N-1 to begin generation, these can be empty, or they can be the beginning of a drum pattern that the network can work off. We also perform the pre-processing steps described above on the input MIDI file to extract the metrics and intensities used when generating.

Once the network begins generation, for the next timestep's input we can roll the input backward by one timestep and append the previous output to the end of the input (feeding the network's last output back to itself.) We also override the intensity values to be that of those extracted from the MIDI file. We then feed this input back into the network to generate the next timestep and repeat this process until the entire track is generated.

We can then combine this track with the original MIDI input to write the final result to disk as a MIDI file.

I separated each section that would have to be implemented into different Python files:

- Drumset.py - Contains the "DrumSetSequence" class used as a Keras sequence dataset to convert data from the saved input file into appropriate inputs for the neural network.
- Model.py - Contains the "Model" class used to implement the Keras neural network model. Provides methods for saving, loading, training, plotting the progress of the network, and predicting outputs.
- Tokeniser.py - Contains functionality for converting MIDI files into token arrays; a form of input that is accepted by the "DrumSetSequence" class.
- Utils.py - Contains utility functions that are used by the other classes (combining two token arrays, normalizing intensities, etc...)
- There is also a Jupyter Notebook file that is used to combine these files together in an easily readable and runnable way.

Results

When training traditional neural networks, it is possible to use the loss value as a measure of accuracy, however for this application of neural networks, the loss does not particularly matter as there is no "solution" or set of solutions for the inputs. The network will never be able to predict the exact next timestep every time. For example, the same drum pattern may be succeeded by multiple different drum fills or sections at different points within the same song. Given that the pattern is the same for each occurrence, there is no way for the network to know what the next timestep will be. As the purpose of this system is to be creative, this may work in our favour as the system should still give an appropriate output in this situation which may not be any of the actual training outputs exactly. This allows the system to turn its own indecisiveness into a form of creativity.

One issue that arouse during the training of the network is the lack of variety of music in the input dataset, an issue described by Collins (2020). The dataset chosen is also exclusively composed of popular tracks starting from around the 1950s. However, songs older than this do not usually contain drums so this may not be a large issue. Most songs are produced by American or European artists and as such music from other cultures are not represented adequately. The results of the system would likely be more varied and more suitable for a larger array of inputs if it was trained on a larger variety of music.

When generating timesteps, the network is sometimes reluctant to generate some of the more infrequently used percussion sounds. For example, if the initial input does not include a snare drum, it will have difficulty generating a snare as an output. In one case, the network generated a snare drum and incorporated it into a pattern, but the snare hit on beats 1 and 3, instead of the more traditional 2 and 4. This is interesting because, according to the intensity values, this is where the system thought that a snare drum would be more appropriate, even though it goes against what most human would think.

Undoubtedly, the abstraction of the input MIDI into just metrics has removed at least some key features and characteristics of the music that would have helped the network to produce a higher quality output. For example, changes in key signature are not extracted at all and may possibly results in a diversion between what a human player would play compared to what the network generated. The fixed eighth note timestep also limits the variation and novelty of the generated output. The timestep could be increased but this would mean the network would take more time to train.

Evaluation

One way to evaluate the creativity of a system is by using the FACE model (Pease and Colton 2011):

Frame

Due to the nature of neural networks, the system cannot explain its output. The closest to an explanation we can get is by looking at the input dataset of MIDI files used in the system. The system cannot explain why it has generated what it has, it is up to the human viewer to infer this information after viewing the output. We should see that the output is similar to existing input examples mixed with the starting input but is specialised to the given input metrics.

Aesthetic

The aesthetic criterion can be described as the extraction of metrics from the input MIDI (intensity, etc…) and subsequent use in training the network. Allows the learning of the link between the metrics and drum events and usually results in an appropriate output given the previous inputs and metrics. Due to the nature of neural networks, all generations and sequences of generations should be appropriate and should not need to be passed through a fitness function.

Concept Generation

The concept can be described as the trained model of the neural network which has learnt to generate an appropriate next timestep given the previous inputs. Also, it can be the method for which the output of the network is converted back into MIDI notes, i.e., confidence cut-off levels. The initial input that the model requires can also be seen as an initial seed that informs the generative process.

Expression of Concept

The expression of the concept can be seen as the prediction of next timesteps from the neural network and the subsequent output MIDI file. The expression in this case is the final combined MIDI file containing the original MIDI notes and the generated drum track using the extracted metrics.

Overall Evaluation

The FACE model requires that each of its components have both processes and subsequent artefacts. In the case of "frame," my system fails as it does not have a method to generate framing information as is required by the frame process criterion. It instead uses a dataset of existing MIDI files that fulfils the role of the framing artefacts; the information used for the remaining ACE of the model. Because of this, the system cannot improve itself or its ability to extract more appropriate metrics from the music that would possibly assist in the aesthetic criterion.

It is also worth noting that there are different methods of evaluating creative software, and that FACE is just one of many methods that can be used.

Conclusions

In its current state, the system produces promising results. With only the intensity information and metrics, the system is able to (most of the time) correctly (at least to a human ear) place snares correctly (on beats 2 and 4), and appropriately insert crash cymbals. The bass drum rhythm will often adjust to the rhythm of the music. The generation can also be customised by adjusting the cut-off and initial input.

The system can also be used as a collaborative tool, that is, a creative collaboration between computer and human. Users can run the system on their own songs and leave it to the system to generate the drum track for them. It should be possible to modify the system so it would run as a plugin within a DAW (digital audio workstation), meaning that users will not have to export their audio into MIDI format before the system can operate on the data. This would also allow the drum track to be directly inserted into the DAW instead of the outputted MIDI having to be manually imported. The system does use a considerable amount of memory, however, it only consumes this memory when it is loaded, so the theoretical plugin can be unloaded when not in use.

Due to the nature of how the intensities are calculated, this system is not able to be ran in real-time and users cannot have the system play along with them. I can see a system such as this one being used to produce song demos quickly. Or perhaps by indie musicians who cannot play (or know how to programmatically input) drums.

References

Collins, N. (2020). *Composition in the Age of AI*. [Online]. Available at: https://composerprogrammer.com/research/CompositionintheAgeofAI.pdf [Accessed: 4 May 2023].

Google (2019). *TensorFlow* [Online]. Available at: https://www.tensorflow.org/.

Huang, T. (2019). *Neural Networks Generated Lamb of God Drum Tracks* [Online]. Available at: https://towardsdatascience.com/neural-networks-generated-lamb-of-god-drum-tracks-45d3a235e13a [Accessed: 6 May 2023].

Magenta (2016). *DrumsRNN.* [Online]. Available at: https://github.com/magenta/magenta/tree/main/magenta/models/drums_rnn [Accessed: 6 May 2023].

Marasmusine (2016). *Dunc's Algomusic by Marasmusine* [Online]. Available at: https://marasmusine.itch.io/duncs-algomusic [Accessed: 7 May 2023].

MIDI Association *GM 1 Sound Set* [Online]. Available at: https://www.midi.org/specifications-old/item/gm-level-1-sound-set [Accessed: 9 May 2023].

Mido Contributors (2017). *Mido - MIDI Objects for Python* [Online]. Available at: https://github.com/mido/mido [Accessed: 7 May 2023].

Pal, A., Saha, S. and Anita, R. (2020). Musenet : Music Generation using Abstractive and Generative Methods. *International Journal of Innovative Technology and Exploring Engineering* 9:784–788. Available at: https://www.ijitee.org/wp-content/uploads/papers/v9i6/F3580049620.pdf [Accessed: 6 May 2023]

Pease, A. and Colton, S. (2011). *Computational Creativity Theory: Inspirations behind the FACE and the IDEA Models.* [Online]. Available at: https://computationalcreativity.net/iccc2011/proceedings/the_cybernetic/pease_iccc11.pdf [Accessed: 3 May 2023].

Raffel, C. (2016). *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD Thesis. Available at: https://colinraffel.com/projects/lmd/ [Accessed: 7 May 2023].

Raffel, C. and Ellis, D. (2014). *INTUITIVE ANALYSIS, CREATION and MANIPULATION of MIDI DATA with Pretty_midi.* [Online]. Available at: https://colinraffel.com/publications/ismir2014intuitive.pdf.

Petit, P. (2021). *The Pyzzicato Project — Piano Instrumental Music Generation Using Deep Neural Networks* [Online]. Available at: https://towardsdatascience.com/pyzzicato-piano-instrumental-music-generation-using-deep-neural-networks-ed9e89320bf6 [Accessed: 7 May 2023].