15-623 Project Report

Juneki Hong
junkih@cs.cmu.edu

April 25, 2016

1 Group

- Juneki Hong

2 Introduction

This project was about trying to model and generate music using deep learning.

This mainly involved finding and preparing training data sets, encoding the data into a format suitable to be read in by the deep learning library (keras), training each model until their respective losses were low, and then producing an encoded output and decoding that output into midi files. Finally, I ran the models for many outputs, and then carefully curated good representative examples to be played at the class concert.

This codebase can be found online at `github.com/junks/music-generation`.

3 Data Preparation

I first found a large collection of midi files online[1]. Once downloaded, I looked for collections of files that all had the same author. I found and separated out collections of Classical music (from which I sampled Tchaikovsky data), as well as pop music (Avril Lavigne, Britney Spears), rock/metal (The Beatles, Metallica), electronic (Daft Punk), and anime (Miyazaki film theme songs).

[1] `http://tinyurl.com/gm4dctt`

3.1 Data Representation

It might have been possible just to feed raw midi data into a deep learning library, but I did not think this would yield good results. For example, I did not want the model to learn how to predict midi on and off events. But rather a single note event with an associated duration.

I wanted to represent music at an abstraction level higher than midi, so I turned to the Allegro representation language.

3.1.1 Allegro

Allegro was an important step to abstract midi data. I wrote my own programs in C++ to call the appropriate allegro library functions to make midi-to-allegro and allegro-to-midi scripts. I then wrote bash scripts that would use these executables, iterating over my midi data collections and converting everything into corresponding allegro files.

3.2 Vectorization

I went even further processing the events listed in the resulting allegro files into vectors. I wrote a program to go through an allegro file and strip away as much meta-data as I could (removing titles, authors, copyright information, song lyrics, etc). I also stripped out tempo change events, stripped out all of the instrument change events, and then crammed all of the note event data into a single midi channel.

For the remaining note events, I went through all of them stripping out all of the identifying text until I was left with a vector of 6 numbers (representing just the pure values: pitch, velocity, duration, etc). I also converted all of the absolute time stamps into relative time deltas (an event at times TW3 and TW4 would be converted into deltas of +3 and +1 respectively). The idea here was that our model would then learn to predict the delay of when to play the next note, rather than predict that there should be some note at some specific absolute time.

What I was left with was a file containing a large list of notes and a few unstripped meta events (I kept any switches between major and minor scales, for example, turning them into their own unique vector of size 6). This would be the final encoded format to be fed into an LSTM to be learned from.

3.3 Pipeline

Apart from the encoding program, I also wrote a corresponding decoding program to undo all of these changes and spit back up the original allegro file. Thus, I was left with a full pipeline that would take midi, convert to allegro, and then convert into vectors; and then afterwards we could convert from vectors, into allegro, finally back into midi.

3.4 Data Representation Future Work

I think it is possible to properly encode the tempo changes and instrument changes that I threw away for this project. If I further work on this project, I would like to get that working as well. The end result would be that our model would then be predicting/playing with a full range of instruments, playing them at variable tempos.

4 Learning

With all of the data prepared, we fed it all into an LSTM model. We sliced contiguous chunks from our training data (100 vectors at a time), and asked the model to predict the next note in the sequence. That is, we provided 100 input vectors and asked the model to predict the next vector.

We experimented a bit, and in the end, we made the slices disjoint. We made separate 100 vector slices that did not overlap with each other, and this was our input training data.

4.1 Architecture

In the end, we built an architecture that took 100 vectors as input, which fed into a size 512 LSTM, which fed into a size V fully connected layer in the end (where V is the size of our vocabulary, every single distinct note event we had seen in training). We then ran a softmax over the output layer to predict the decoded output note.

4.2 Decoding

We randomly sampled 100 contiguous vectors from somewhere in the training data as a starting initial input. We then fed it through the LSTM to decode

the next output, we then put that output as part of the input, feeding that result back into the LSTM for the next note, and so on.

4.3 Future Work

There are endless variations to this deep learning architecture that could have been tried. We stuck to a relatively simple architecture with a single hidden layer and a single fully connected output layer.

One thing we can try is to stack one more additional LSTMs after the first one, to see what might happen. Another thing to try is to map the length 6 vectors into a higher dimension (such as 300) with a fully connected layer in the beginning.

Another hindrance to this project was the lack of large scale computing resources. The server I usually had access to (that had GPUs), went down the week I ran this project, so all models had to be small enough to be run comfortably from my laptop.

With larger resources, we could run with much more data and try running much larger/complicated architectures.

5 Criteria to Evaluate Project, Assessment

All of the above work to get everything working together was done by me. This includes scripts/programs to fit into an end-to-end data pipeline, as well as the experiments and final architecture design of the LSTM.

In the end, the project worked and produced music, even though it was not able to produce music in real time. The weird effects of the LSTM slowly losing coherence and recognizability was an interesting side effect in itself. Perhaps interesting to the audience, listening not only to the slow descent into madness, but also the somewhat familiar notes and sequence fragments that might occasionally still get played.

I believe the project has been successful, at least as an initial foundation for more complicated modeling and generation. Some of the future work has been described above.