1-1-2018

RNN-Based Generation of Polyphonic Music and Jazz Improvisation

Andrew Hannum
University of Denver

RNN-Based Generation of Polyphonic Music and Jazz Improvisation

Abstract

This paper presents techniques developed for algorithmic composition of both polyphonic music, and of simulated jazz improvisation, using multiple novel data sources and the character-based recurrent neural network architecture *char-rnn*. In addition, techniques and tooling are presented aimed at using the results of the algorithmic composition to create exercises for musical pedagogy.

Document Type

Thesis

Degree Name

M.S.

Department

Computer Science

First Advisor

Mario A. Lopez, Ph.D.

Keywords

Artificial intelligence, Generative music, Jazz improvisation, Machine learning, Neural network, Recurrent neural network

Subject Categories

Artificial Intelligence and Robotics | Computer Sciences | Music Pedagogy

RNN-based generation of polyphonic music and jazz improvisation

A Thesis

Presented to the Faculty

of the Daniel Felix Ritchie School

of Engineering and Computer Science

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Andrew Hannum

November 2018

Advisor: Mario A. Lopez

Author: Andrew Hannum
Title: RNN-based generation of polyphonic music and jazz improvisation
Advisor: Mario A. Lopez
Degree Date: November 2018

Abstract

This paper presents techniques developed for algorithmic composition of both polyphonic music, and of simulated jazz improvisation, using multiple novel data sources and the character-based recurrent neural network architecture *char-rnn*. In addition, techniques and tooling are presented aimed at using the results of the algorithmic composition to create exercises for musical pedagogy.

Acknowledgements

This project would not have been completed without the close guidance and assistance of Dr. Mario Lopez (professor of Computer Science). I cannot give enough thanks for his openness, availability, insight, and inquisitiveness throughout the entire process. It is also thanks to his constant encouragement and advocacy that this project resulted in a Master's thesis. There are no words to encompass the degree of his contributions to this project, and to my career and development as a computer scientist. Thank you, Mario.

The jazz improvisation aspect of this project would not have been possible without generous donations of transcriptions from two talented and accomplished musicians:

Benjamin Givan, author of *The Music of Django Reinhardt* - Ben transcribed over 200 Django Reinhardt solos as part of his own research, and the transcriptions have since made their way around the gypsy jazz online community. While considering the possibility of simulating jazz improvisation, and discovering a lack of appropriate input data, I reached out to Ben directly thinking that his corpus of transcriptions might provide the input needed, but with no real expectation of any kind of response. To my great delight, he responded quickly and enthusiastically, and his transcriptions provided the foundation that was needed to get that part of the project off of the ground. Without his generosity I would likely have given up on the jazz improvisation simulation. Thank you, Ben.

Denis Chang, founder of DC Music School - Denis makes a living teaching and playing gypsy jazz, and sells his jazz solo transcriptions for money via his online music school. I had the pleasure of staying with Denis and studying gypsy jazz guitar with him at his home in Montreal. At that time, I had a foundation for the jazz improvisation data set but the results were somewhat unimpressive. Looking to increase to size of the data set, I asked Denis if he would be willing to share any of the original files for his transcriptions. This was a hard question to ask since, unlike Ben whose transcriptions were made for academic purposes and were already freely available, Denis's transcriptions are commercial and support his ability to make a living. Nevertheless Denis responded with interest, and giving me the original files of the transcriptions demonstrated a trust that I am immensely grateful for. Without his donations the results of the simulated jazz improvisation would likely have remained stale and uninteresting. Thank you, Denis.

Contents

List of Figures

Chapter 1

Introduction

1.1 History of Algorithmic Composition

Algorithmic composition is the process of composing music by automated systems or formal sets of rules, rather than by human intuition or inspiration. Implementations of algorithmic composition can be found dating back centuries, far in advance of computer systems - a notable early example from the 18th century is *Musikalisches Würfelspiel*, a system attributed to Mozart [2] of using dice to randomly generate music from pre-composed options. With the advent of computers, algorithmic composition systems have become more commonplace, and have expanded greatly in their variety and complexity.

Many of these early systems, including computer implementations, could be generally lumped into one of two categories: stochastic systems, and formal grammars. Stochastic systems, like Mozart's dice game, make use of randomness and probability to generate their compositions. Formal grammars are rule-based systems that follow a strict set of steps for composition, though these are often combined with stochastic approaches to create hybrid systems.

In recent years, a new category of algorithmic composition has been developing - that of systems based on artificial intelligence techniques. Artificial intelligence systems share characteristics with both stochastic and formal systems, but they differ fundamentally in that their compositional rules are "learned" by the use of machine learning techniques. Usually, learning architectures are devised that take existing music as input in some form, and learn to mimic the features of the learned music in their output.

Even within the category of artificial intelligence systems, there are many varieties of implementations using different learning architectures, different music data sets, and different compositional goals [4] [7] [8] [10]. Such systems display a wide range of success or failure, with some systems producing unmusical results, some systems producing impressive and interesting music, and many systems producing results somewhere in between.

1.2 Motivations of This Project

Much of the motivation of developing algorithmic composition systems is simply to create interesting or entertaining music. However, there is potential for such systems to be used for other purposes which might lend themselves better to systematic approaches. One such potential use is the creation of exercises for musical pedagogy.

When learning to play an instrument, students will inevitably have to learn fully (human) composed pieces of "good" music. However, for many serious students of music, much time is also spent on pieces of music which have been composed not for entertainment value or beauty, but for their effectiveness in teaching specific skills.

There are books full of short pieces of music known as *études* whose purpose is entirely pedagogical, usually developed for learning specific techniques for an instrument, or a certain style of music. These musical études are often much more

mechanical and rule-based than normal music, and so perhaps such pieces would lend themselves well to algorithmic composition techniques.

Creating a formal system for composing musical exercises is tempting but previous attempts have shown a tendency to often produce somewhat unmusical-sounding results [3]. This project attempts to use an artificial intelligence system that has already shown a promising degree of musicality in its compositions to create exercises for musical pedagogy which sound reasonably musical.

Chapter 2

Background

2.1 Music in Audio vs Abstractions

The ultimate experience of any music is that of pressure vibrations, propagated most often through the medium of air, perceived by the human brain via inner-ear membranes that stimulate neural signaling via auditory nerves. However, music conceptually exists on a spectrum, with one side of the spectrum being the material reality of the sound waves, and the other side of the spectrum being the theoretical abstraction of music in the form of notes, harmonies, rhythms, and the language used to communicate about those abstractions. Both sides of the spectrum are music, but for the purposes of discussion it is important to distinguish between the two different senses of music.

Instruments, from this perspective, are a means of translating from the abstract concepts of music (as it might be written on sheet music for example) into the objective reality of sound waves. Sheet music is a representation of abstract musical concepts, whereas a recording (or a sound wave itself) is a representation of the objective reality of the final form of music before being experienced.

2.2 Digital Representation Format for Physical Audio

The Waveform Audio File (WAV) format is one of the rawest forms of audio data structure used in digital applications. It is, essentially, just an array of time slices containing measurements of the amplitude of the sound wave. This is the initial file format used for most forms of audio recordings.

2.3 Music Theoretical Concepts

Pitch Refers to the perceptual property of a sound as it related to its fundamental frequencies - whether a sound is perceived as "higher" or "lower".

Note Designation for a specific pitch and duration of sound. In modern music theory, notes are divided into conceptually equidistant pitch classes where a note is in one of 12 pitch classes: A, A#/Bb, B, C, C#/Db, D, D#/Eb, E, F, F#/Gb, and G.

Beat The basic unit of time used to indicate rhythm.

Time Signature A measurement used to subdivide and group beats (esp. in a written or visual notation system). It indicates how many beats occur per measure, the divisions of each beat, and how those beat divisions should be notated. For example a time signature of 3/4 indicates a grouping of 3 beats per measure where each beat is notated with a quarter-note.

Measure One unit of beats grouped based on time signature.

Step, Tone, Interval The distance between two notes within the set of standard pitch classes is known as the interval between them. By convention, the distance between two immediately adjacent pitch classes is a half-step or semitone, with the next largest interval being a whole-step or whole tone.

Octave Two notes with frequencies in a ratio of any power of two belong to the same pitch class, and are called octaves. For example, 220 Hz and 440 Hz are octaves, both with the pitch class A.

Chord A set of notes played simultaneously. Chords have a deep set of music theory conventions for naming, where the name or symbol for a chord is usually based on the lowest pitch present in the chord, and the intervals present between the set of pitches.

Key Signature Refers to a subset of the possible pitch classes which are emphasized in the piece. Most often a key signature is a subset of 7 pitch classes with a specific interval relationship known as the diatonic scale (two half steps separated from each other by either two or three whole steps).

For a more detailed explanation of relevant music theory concepts please refer to *Music in Theory and Practice* [1].

2.4 Digital Representation Format for Abstract Music

Musical Instrument Digital Interface (MIDI) is a widely-used format for representing abstract musical data consisting of notes, chords, rhythms, keys, time signatures - essentially the data at the opposite end of the music spectrum from sound waves.

MIDI consists of a stream of messages, the most important of which to this project are the Note-On and Note-Off messages. With these messages, most of the musical content of a performance or of a written music piece can be represented digitally. When mapped as pairs of On/Off intervals, such data can be visualized as a piano roll - a more intuitive way of understanding the data that MIDI messages are representing, and often a more intuitive way of working with the data (see Figure 2.1).

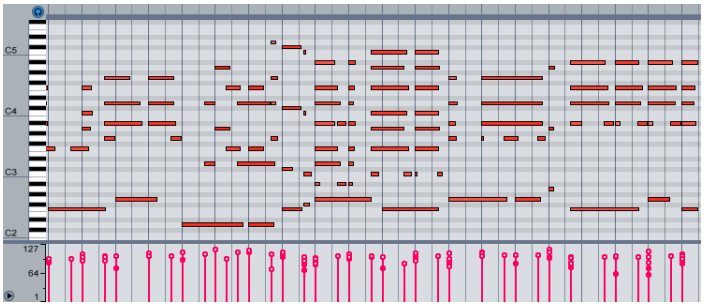


Figure 2.1: A MIDI piano-roll visualization

MIDI by itself can not be directly listened to, it requires the use of digital instruments to convert the abstract music representation to sound waves. There are many such digital instruments available with varying levels of quality and realism.

2.5 Artificial Neural Networks (NNs)

Artificial neural networks (NNs) are a wide class of machine learning architectures that resemble the networks of neurons observed in biological neural systems.

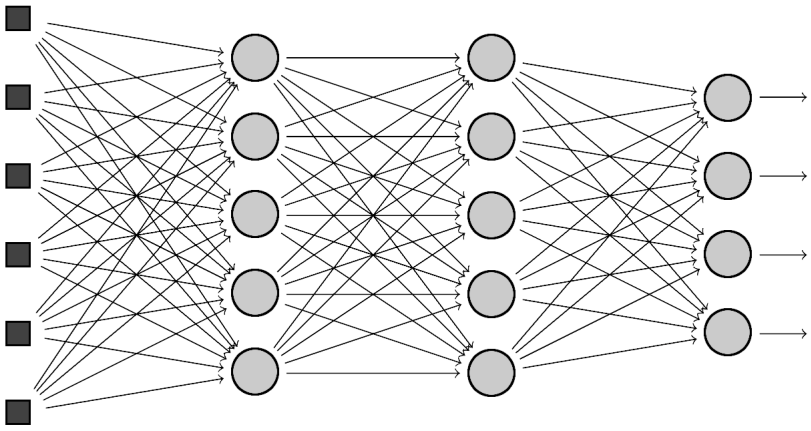


Figure 2.2: A standard artificial neural network architecture

They are characterized by groups of connected nodes arranged in layers (see Figure 2.2), where each node in an active network has a scalar "activation" value that is calculated based on inputs to that node. The activation values propagate

forward through the network to transform the input into some output, dependent on the connections between the nodes and their activation states.

In most cases, there is at least one input layer, some number of intermediate "hidden" layers, and an output layer. Each node has a number of input connections from some other nodes, usually in preceding layers, with each connection having a variable weight, that the individual node combines via some mathematical function to calculate that node's activation value (which is, in turn, used as input to some number of other nodes in other layers). A simple example of a node activation function would be a threshold, where the node activates only if its aggregate inputs exceed some set value.

In a broad sense, artificial neural networks can be thought of as learning to approximate a function, where some set of inputs are mapped to some set of outputs. This is done primarily by adjusting connection weights until the input produces the correct output.

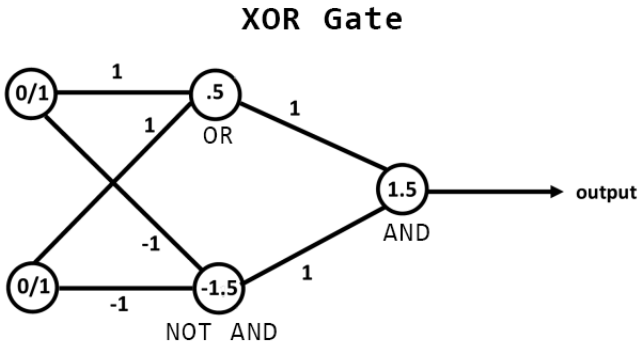


Figure 2.3: A simple NN including XOR, NOT AND, and AND gates

To help visualize how neural network nodes and connection weights can be used to approximate simple functions, see Figure 2.3: a neural network which reproduces the functionality of three simple logic gates. Each input node sends an activation value of either 0 or 1. That activation value is multiplied by the connection weights

and summed in the subsequent hidden layer nodes. The value seen on the hidden layer nodes is a threshold - only if the sum of its inputs exceeds that threshold will the node send it's own activation value of 1, otherwise it sends 0.

Learning to approximate a function (a process called "training") occurs by presenting the input and output layers of the network with known examples of correct input/output mappings, and calculating the changes that need to be made to the inter-node connection weights in order for the input layer to ultimately produce the desired output in the output layer.

The aggregate difference between the output produced by the current state of the network, given some input, and the theoretically correct output of the network, given the same input, is known as the "loss" or "error". While training the network on known correct input/output pairs the loss, called the "training loss", is used to incrementally update the connection weights of the hidden layer nodes to more reliably produce the desired output via a process known as "backpropagation".

By making incremental corrections to the connection weights between nodes, over training with many correct input/output pairs, the expectation is that the network learns to generalize strategies or functions that will reliably produce the desired output given any input.

The final ability of the network to map an input to the correct output is usually measured by maintaining some known correct input/output pairs (called the "validation set") aside from the training data set, and testing the trained network for the loss associated with this untrained data (called the "validation loss"). Thus the validation loss is used as an indicator of how successful the network was in its training - a measure of how effectively and reliably it can produce the desired output from a never-before-seen input.

For a more detailed explanation of artificial neural networks please refer to *Fundamentals of Artificial Neural Networks* [6].

2.6 Recurrent Neural Networks (RNNs)

A recurrent neural network (RNN) is an artificial neural network that attempts to make use of the sequential nature of some kinds of input by adding to hidden layer nodes additional "recurrent" connections from themselves to themselves in a future "time step". See Figure 2.4 for a simplified example, where each layer of the network consists of a single node with a recurrent connection to itself.

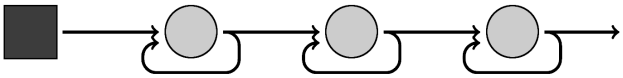


Figure 2.4: A simplified recurrent neural network architecture

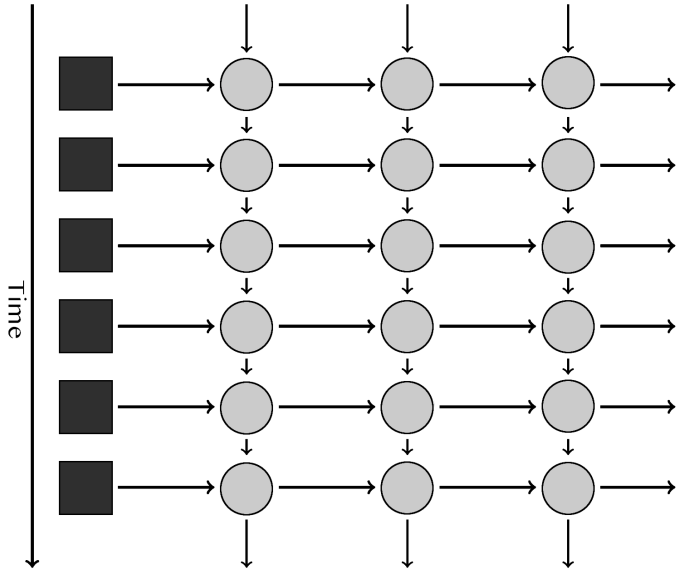


Figure 2.5: A RNN unrolled through time

These networks are trained on sequences of input by considering one step of the input sequence at a time, and then "unrolling" the network for some sequence length (see Figure 2.5) - a process where the nodes in the network at one time step are used as inputs to the same nodes in the next time step by way of their recurrent connections. After unrolling the network for the given training sequence

length, backpropagation happens over the entire unrolled sequence (a technique called "backpropagation through time" or BPTT) to recalculate weights for both the inter-layer connections, as well as the recurrent connections for each node.

These style of networks have been shown to produce much better results for many types of sequential inputs than non-recurrent neural networks. RNNs have been anecdotally described as learning to approximate a state-machine, where standard NNs learn to approximate a function.

2.7 Long Short-Term Memory (LSTM) RNNs

Long short-term memory is a modification to RNNs whereby hidden layers are made up of LSTM units containing an "input gate", an "output gate" and a "forget gate" (see Figure 2.6). Each gate has essentially the same functionality as a node in a standard RNN, but by grouping them together into units the use of the gates allows the units to "remember" information for an arbitrary length of time. At each time step, given the activation input from preceding units and the activation of itself in a prior time step, the LSTM unit "decides" whether or not to forget the information that it is currently "remembering".

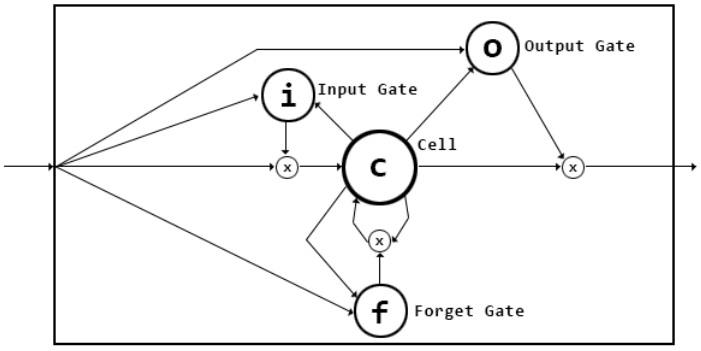


Figure 2.6: A long short-term memory unit

LSTM networks have been found to be effective at predicting time series where unknown or irregular time intervals occur between important events. In a loose sense, LSTM networks have an explicit memory system that standard RNNs do not.

2.8 char-rnn

char-rnn[1] is a LSTM RNN implementation by Andrei Karpathy that learns a character-level predictive language model from a large data set of input text.

The primary aim is that, given an input sequence of characters, the network learns to predict the next character in the sequence. The most interesting aspect of the project is that once it has been trained to predict the next character in a sequence, the network can be used to generate texts "in the style" of the training texts. This is done by having the network predict the next character after an input sequence, adding that character to the end of the sequence, and using that as the new input sequence, ad infinitum.

Given a sufficiently large set of training data, the results of such generation are found to be impressively coherent, especially in short sequences, but even with some larger structural characteristics of the input text. The resultant pieces when trained on something like the English language could not be confused for true literacy, but they seem to capture something characteristic about the "style" of the input.

For example of char-rnn's output after being trained on the collected works of Shakespeare, see Figure 2.7.

[1] https://github.com/karpathy/char-rnn

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Figure 2.7: Example output from char-rnn trained on Shakespeare

2.9 ABC Notation

ABC notation[2] is a simple textual representation format for abstract musical data. It uses simple lettering to represent notes, along with modifiers for pitch, ornamentation, durations, and other details. It is designed to be limited to the ASCII character set.

For a short example of ABC notation and the resulting music, see Figure 2.8.

```
X:1
M: 4/4
L: 1/8
K: Emin
D2|EBBA B2 EB|B2 AB dBAG|
```

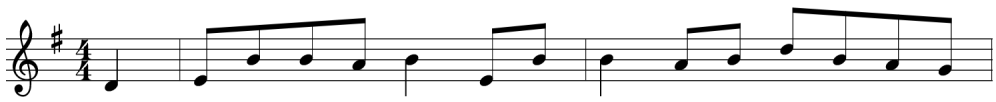


Figure 2.8: Example of ABC notation and the resulting sheet music

2.10 Lisl's Stis

Lisl's Stis is a project by Bob Sturm that uses a large, freely-available corpus of ABC-notated Irish Folk music[3] as training input into the *char-rnn* network. After training, the network is made to generate text "in the style" of the ABC-notated Irish Folk music.

The music that results from letting the trained network freely generate are often impressive, in part due to their cleanliness (which is perhaps just a factor of the notation system used) but also in their subjective adherence to the Irish Folk style. Some of the results could reasonably be confused with actual Irish Folk melodies, to an inexpert listener.

[2] http://abcnotation.com/

[3] https://thesession.org/tunes

The pieces also have entertaining value resulting from some artifacts of the format of ABC notation - specifically the pieces have auto-generated titles which are reminiscent of the titles of real Irish Folk tunes. One such title gave the project its name: "Lisl's Stis".

For an example of the output from Lisl's Stis, see Figure 2.9.

```
T:Lat canny I. the dlas.
M:C
L:1/8
Q:1/2=100
K:D
A>A|:F>DEA|F2dF|A/F/A/B/ AF|G/E/E FD |DDDG|Edec|
defd |eged|fdgd|dcd2||
e|g2ef gef(e/c/)|ddfe fdAA|F3 A c4|efef g{e}d4 |
gfga afgf|eggb ad'eg|fgdB edAB|BedA BABg|fdde ddd:|
```

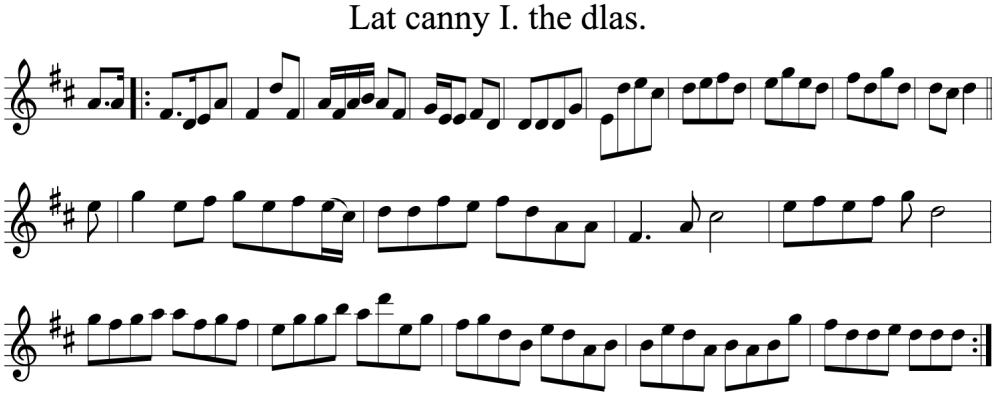


Figure 2.9: Example output from Lisl's Stis

Chapter 3

New Contributions

3.1 Leveraging MIDI to Access Other Data Sets

Given that the results of the original Irish Folk music *char-rnn*/ABC project appear promising, it is tempting to try applying the same techniques to other styles of music. However, the size of the freely available corpus of ABC-notated Irish Folk melodies appears to have been a lucky break. ABC notation is not used extensively enough for there to exist any other comparably large corpuses of ABC-notated music to try. Thus, further experimentation in the usage of these techniques is difficult due to a lack of adequate data sets.

This project addresses this lack of alternative data sets by leveraging the availability of sufficiently large corpuses of MIDI-notated music (a much more commonly used format) and freely available format-conversion software to convert such corpuses into the appropriate ABC-notated format. The conversion process, though smoothly automated once fully implemented, comes with many of its own difficulties and caveats that are detailed in the methods.

3.2 Application of Same Techniques to Polyphonic Data Set

The use of MIDI-notated corpuses opens the doors to many potential new data sets, however, in the interest of progress it seemed important that experiments be run on a corpus which was in some way fundamentally different than the original Irish Folk corpus. For example: music that is either more varied, or more complex, than the relatively simple and highly-regular Irish melodies. At the same time, an ideal corpus would be one which differs from the Irish Folk corpus in only that one fundamental way and not in many ways at once (i.e.: one that is not so drastically different that there can be no expectation of comparison).

The first corpus used was a collection of freely available MIDI-notated classical guitar pieces[1]. Without needing to exhaustively analyze each piece of music in this corpus, the fact that it is composed entirely of classical guitar pieces from the common practice period provides many expectations about the regularity of the pieces that make it a reasonable fit for the above requirements. Pieces are most likely in a similar musical range, and in similar musical keys, due to being written for the same instrument. In addition, because all pieces share a similar style of music, one with a reasonably characteristic "sound", it is likely that similar musical motifs can be found in between the pieces. In these ways the classical guitar corpus is analogous to the Irish Folk music corpus - regularity of musical range, musical keys, and stylistic motifs.

The classical guitar corpus differs from the Irish Folk corpus in that its music is generally more complex, and most significantly that it consists almost entirely of pieces of music that make heavy use of homophony (music with chords) and polyphony (music with more than one voice playing simultaneously). This increase

[1] http://www.classicalguitarmidi.com/

in complexity and the use of musical structures that are absent in the Irish Folk corpus provides a contrast that, combined with the similarities mentioned, make the classical guitar corpus a good candidate for pushing the limitations of the *char-rnn*/ABC technique in a controlled way.

3.3 Contextually Constrained Improvisation

After obtaining satisfying results with the classical guitar data set by directly replicating the techniques from the Irish Folk project, another musical corpus was chosen with the more ambitious goal of developing a new technique: using the RNN's priming features to simulate contextually constrained improvisation.

The details of this technique will be discussed in the methods section, however it's worth noting here that such a technique requires that the input to the RNN contain both the improvised music and the context constraints in which it was improvised. This is simply to say that a corpus full of digitized improvisation would be worthless if those improvisations did not also contain the context (in this case the harmonic context, or chord progressions). Some other approach might try to introduce the idea of context as an input by altering the architecture of the NN, however in this case no architectural changes were made - harmonic context was simply added as an additional feature of the notation.

3.4 Gypsy Jazz Improvisation

Jazz music is an obvious choice when looking for examples of well-structured contextual improvisation, in the form of improvised jazz solos: pieces of single-instrument music which are improvised in real-time around accompaniment with a set rhythm and known chord progression structure. Furthermore, although jazz has become more free-form as it has developed over time, and "playing outside" has

become more acceptable and expected within improvised solos, historically older forms of jazz have the benefit (in this case) of being more regularly structured, and adhering closer to a set of "rules" regarding "correctness".

Gypsy jazz in particular is an older style of jazz pioneered by guitarist Django Reinhardt, which presents a number of features that (as with the classical guitar corpus) lend themselves well to maintaining a degree of regularity between pieces. As with the classical guitar corpus, such pieces are all exclusively played on guitar, and the musical keys used are often chosen to work well on the guitar (although other forms of jazz may also feature such regularity, but in a form that favors horn-centric keys such as Bb). In addition, the vocabulary of phrases in gypsy jazz is more uniform and limited than in jazz as whole. In the gypsy jazz community there is heavy emphasis on learning and employing stylistic licks and phrases (in an effort to sound like Django), and as a result gypsy jazz has a stronger characteristic style than most other forms of jazz.

That being said, attempts to find a sufficiently large corpus of well-structured digitized jazz solos, let alone gypsy jazz solos, proved impractical - especially given the above-mentioned requirement that they include a digitized version of their harmonic context. There are many examples of MIDI-notated jazz solos, but the vast majority lack any kind of harmonic context, or in some few cases have difficult-to-parse and non-uniform ways of representing the chord progressions (MIDI does not have a built-in way of representing accompanied chord progressions beyond explicitly notating them using polyphony). As a result, for this data set leveraging an existing MIDI corpus was not an option.

In the end, a corpus was obtained by generous contributions from Ben Givan[2] and Denis Chang[3], who have both spent significant amounts of time meticulously

[2]https://www.skidmore.edu/music/faculty/givan.php

[3]https://www.dc-musicschool.com/

transcribing gypsy jazz solos for their own research or commercial purposes. The original transcriptions were received in the format of music typesetting programs Sibelius[4] and Finale[5], converted by various means to ABC, and combined with a custom representation of harmonic context, which will be discussed later.

Improvisation was simulated using the network trained on this data set by interrupting the output generation at appropriate intervals, "injecting" the appropriate harmonic context cues into the output, and then continuing output after the injected context cues - as discussed in the methods section.

[4]http://www.avid.com/sibelius

[5]https://www.finalemusic.com/

Chapter 4

Methods

4.1 Some Methodological Considerations

At all stages in the process, some heuristics guided the decision making process, which should be mentioned before detailing the specific methodologies. These heuristics are vague and somewhat unjustified, and not necessarily backed by concrete experimental results or theoretical rigor. They are rules of thumb based on general observations and abstract analogous reasoning about neural networks and the way that they learn. Despite the lack of rigor, some heuristics were needed and these were some of the main guiding assumptions. It is always assumed that:

- Simpler (less verbose) input is desirable because it will be learned more easily than complex (more verbose) input, when representing the same data.
- More regular input is desirable because it will exhibit higher predictability, and thus be more easily learned by the network, and also because it will produce more consistency in output than less regular (highly varied) input.
- Less repetitive input is desirable because it will reduce the chance of output getting stuck in repetitive loops, and because repetition is an easy way for the

network to "cheat" at predictability while missing the intended content (it is also implicitly assumed that the desired content is not highly repetitive).

- Input data which is not directly related to the desired output is a waste, because the attempt to learn it will occupy some amount of network resources that could otherwise be used to learn the desired output.

4.2 Sorting and Rejection of Bad Data

A first pass was made of the raw classical guitar MIDI files to improve regularity of what would ultimately be input into the neural network. MIDI files were sorted by time signature into groups of common binary time signatures (where the numerator is divisible by 2, e.g.: 4/4, 2/4, 8/8, etc.), common ternary or compound time signatures (numerator divisible by 3, e.g.: 3/4, 6/8, etc.) and otherwise unusual time signatures. The binary time signature group was the largest by far, the only group large enough to appear sufficient for training, so it was the group that was used for most experiments.

Additionally, at this stage MIDI files were removed if they appeared to be mostly or entirely empty of substantive musical data, or were otherwise deemed bad (erroneous MIDI data for example). This consisted of a mostly insignificant portion of the overall data set (less than 1%).

4.3 Splitting and Cleaning Raw MIDI

The MIDI files being used were not all created by the same author and so exhibited a wide variety of formatting inconsistencies between files in their usage of headers, track layouts, instrument choices, timing, and MIDI message conventions (e.g.: the use of SYSEX messages, program changes, note on/off lengths, etc.). To improve regularity a few techniques were used.

First, files were parsed manually using a custom Ruby[1] script and the *midilib* library and transformed as follows:

1. Separate tracks were identified (some MIDI files used separate tracks to represent separate voices) and recombined into a single track. Any tracks which were not guitar (metronome or other instruments for example) were removed.

2. The single remaining instrument track was stripped of all MIDI events other than note-on and note-off events.

3. All existing headers were stripped from the file and replaced with a consistent set of headers: track name, instrument, and initial program change (PC) message (specifies the patch used to emulate the instrument sound).

4.4 Increasing Data Set Size

At one point, after a few experiments with the original data set, an attempt was made to increase the size of the available data set by transposing MIDI files to other musical keys. The assumption being that a piece of music in the key of G major, for example, would still be a valid example of desirable output if it were in the key of A major, or D major. Such additional examples can be easily produced by transposing all notes in the file up or down by the same interval.

In the interest of not producing too much artificial variation a limited number of transpositions were used - all files were transposed into the major keys of A, C, D, E, F and G (or the relative minor keys).

It was at this stage of the process, while still in raw MIDI format, that transpositions were created, again using a Ruby script and the *midilib* library.

It was unclear in the end whether this artificial increase in data set size improved the results (especially lacking a metric that could measure such an improvement),

[1] A widely-used programming language

but subjectively it did seem to provide some more variation in the output. There is no quantitative evidence of this, however.

4.5 Cleaning MIDI with MuseScore

As the last step of cleanup on the raw MIDI level, all MIDI files were imported into MuseScore[2] and directly re-exported again as a MIDI file. MuseScore contains a robust set of logic rules for importing and interpreting MIDI files into sheet music format. At this stage, the sheet music representation is irrelevant, however, because the import system is designed to work with inexact MIDI data (even MIDI files of human performances) the interpretive logic was found to be immensely helpful in producing better regularity in note event timing and removing subtle errors or inconsistencies, while leaving well-formed data unchanged.

4.6 Converting MIDI to ABC Notation

Conversion from MIDI files to ABC notation was achieved simply by using the *midi2abc* binary from the *abcmidi*[3] package. Pieces in ABC notation were stored simply as text files.

4.7 Cleaning ABC

Despite efforts made to ensure uniformity between MIDI files, there remained some irregularities and artifacts after conversion to ABC notation. Rather than track down the source of these artifacts, a Ruby script was used to clean the resulting text files.

---

[2] https://musescore.com/

[3] https://github.com/leesavide/abcmidi

Some ABC files still contained an additional empty second voice, or the main voice was labeled as a second voice, so all extra voices were removed and remaining voices were labeled as voice one.

The conversion process also produced some text components that were deemed unnecessary, such as comments and superfluous line breaks, which were removed.

Eventually, in keeping with the heuristic that any input data not directly related to the desired output is a waste, the titles of the pieces were also removed or made uniform at this stage as well (both approaches were tested, without resulting in significant differences).

4.8 Training RNN on Classical Guitar Data Set

The resulting ABC notations files were concatenated together into a single text file, and fed into the *char-rnn* network for training. The *char-rnn* code exposes a number of variable parameters that affect the training and network architecture. The parameters that were deemed relevant are:

rnn_size Number of nodes in the LSTM internal state. For all experiments run, this was set to 512 (at the recommendation of the author of *char-rnn*).

num_layers Number of layers in the LSTM. The recommended number of layers is 2 or 3. One experiment was run with 2 layers, but the results were subjectively worse, and all following experiments were run with 3 layers.

dropout Dropout for regularization, used after each RNN hidden layer. This is a percentage of nodes in the NN which are randomly not trained during a training pass, in order to prevent over-fitting. The majority of experiments were run with dropout of 0.5. One experiment was run with a dropout of 0.75, but it did not seem to significantly affect the results.

seq_length Number of timesteps to unroll for. This is essentially how far back in "time" should the network calculate loss function, in this case it is the length of the character sequence that the network trains on during each round of training. For the classical guitar corpus, values of 100, 150, and 250 were tried. There was subjectively no significant difference between 150 and 250, however larger values overall seemed to produce better results, so the highest value of 250 was used for most experiments. For the gypsy jazz corpus, smaller values appeared to produce better results - the value used in the final examples was 100.

At regular intervals during training, *char-rnn* saves a checkpoints of the state of the network to a file, and labels it with the validation loss of that state of the network. As mentioned before, the validation loss is assumed to be an indicator of the quality of the state of the network.

The network was trained until the checkpoints produced appeared to hit a relative minimum in their validation loss numbers.

4.9 Let Network Generate Results from Scratch

The checkpoint with the lowest validation loss from each training run was selected, and then used to generate samples either freely, or in the case of some experiments, with priming text intended to produce a specific output (for example, in attempts to produce outputs in certain musical keys, or outputs that were elaborations of existing musical pieces).

For experiments on entirely freely generated samples, a priming text was still provided, but was limited to generic ABC notation headers that were uniform across all pieces, and would not bias the network towards any specific output.

For the sampling process, *char-rnn* takes as an input parameter the *temperature* of the sampling. Temperature is a value from 0 to 1 representing the degree of randomness to apply to sampling from the network's distribution of "best guesses" for the next character in the sequence. A value of 0 will always choose the network's "best guess", usually resulting in an infinite repetition of some small snippet of text, while a value of 1 will result in near-randomness. Through experimentation, a temperature of 0.7 was deemed to produce the results with the best balance of producing correctly-formed ABC notation, while also producing interesting musical content.

Some amount of incorrectly-formed ABC was still usable, as the software used to convert ABC notation to MIDI has a degree of fault-tolerance.

4.10 Post-Processing Network Output

Given that the network was trained on a multi-piece concatenated file, with full ABC notation headers, the sampling stage would often produce multiple pieces in ABC notation, one after the other. A Ruby script was written to process the text samples and split them into separate pieces.

Additionally, although at this point the hope was always to have produced results consisting of valid ABC notation, there was still always a chance of artifacts, and so during post-processing some cleanup was done as well - mostly re-introducing necessary headers if they were missing. It was found that often the network would skip including the necessary voice header "V: 1", for unclear reasons.

4.11 Converting from ABC to MIDI

Having a number of ABC notation samples that were generated by the network, the results were converted from ABC notation back into MIDI using the *abc2midi*

binary from the *abcmidi* package. This would sometimes produce errors if the ABC notation was not well-formed, but the *abc2midi* program generally fails gracefully, by correcting errors where it can, so the majority of samples pieces were successfully converted to MIDI format.

4.12 Rendering MIDI Results to Sheet Music

Achieving a substantial degree of accuracy in music typesetting, especially for multi-voice polyphonic music, is a large problem in itself, and would be impractical to try to solve fully for the purposes of this project. For the sake of ease MuseScore was used again to render the MIDI files to sheet music, to make use of its robust MIDI import capabilities.

Despite the relative ease and initial quality of the results, for the sake of producing some better visual results and consistency in the output, some extra tools were developed as part of this thesis for use with MuseScore: a MIDI import options XML file, and a QML plugin to move rests so that they would not overlap with notes on the staff.

4.13 Listening to MIDI Results of the Classical Guitar Corpus

MIDI files can usually be listened to directly by opening with programs included in most modern operating systems. Standard MIDI instrument banks are included in the operating system and the rendering is done automatically. At this point, the results were ready to be listened to and evaluated for musicality.

For results that were deemed most interesting, and extra step was taken to manually render the MIDI file to audio using more professional-sounding instruments than the ones included in an operating system sound bank.

Rendering for the classical guitar corpus was done using the Native Instruments Kontakt[4] orchestral string sound banks. Though the music was supposed to mimic guitar music, the orchestral strings produced a more interesting final result.

4.14 Priming the Network to Produce Specific Target Pieces

Attempts were made to use the "priming text" feature in *char-rnn* to produce a targeted output. With the classical guitar corpus, this was limited to simple attempts at elaboration of a musical idea. The network was primed with an ABC notation sequence of 8 measures, and then sampled for its "guess" at what the next 8 measures would be. Results were rendered to sheet music and audio in order to judge the subjective similarity and coherence of the priming piece and the elaboration.

4.15 Converting Proprietary Notation Formats to ABC Notation

Gypsy jazz transcriptions were received from Ben Givan in the form of .musx (Finale) files, and from Denis Chang in the form of .sib (Sibelius) files. With no means of converting directly to ABC notation, both sets of files were converted first to MusicXML as an intermediary step. Batch conversion is possible directly in Sibelius, and in Finale, via their own means. Pieces were converted from MusicXML to ABC format using the *xml2abc* python script[5].

[4]https://www.native-instruments.com/en/products/komplete/samplers/kontakt-5/

[5]https://wim.vree.org/svgParse/xml2abc.html

4.16 Cleaning ABC Derived from XML

The results of conversion using *xml2abc* produced far messier results than the MIDI to ABC conversion, including text with no musical content. For this reason a much more elaborate Ruby script was used to clean the results.

As before, all headers except the necessary ones (voice, title, program change) were removed, and all voices were combined into a single voice. In addition, the following cleanup steps were also necessary:

- Remove quoted parentheses
- Remove numbers and special characters followed by exclamation points
- Remove cash signs
- Ensure spaces appear after quotes
- Move all quotes symbols to their own line
- Remove line numbers
- Move chord symbols to their own line
- Add measure bar lines for empty chords
- Remove quoted caret characters
- Remove quoted backslashes
- Remove excessive repetitive rests

4.17 Development of Symbolic Representation of Harmony

ABC notation does not have a built-in way of representing harmony, but it has a notation for displaying text above the music staff. This has, by convention, been used as a way of displaying chord symbols, and the conversion from Sibelius or Finale, to XML, to ABC, resulted in rough chord names being prepended to the measures in the form of this double-quoted text.

For the sake of concision in number of characters used, simplifications were made to chord extensions based on personal understanding of jazz harmony and improvisation. These changes apply only to jazz and represent the commutability of some differently notated chords. The following changes were made to chord symbols:

- "dim" → "o"
- "m7b5" → "0"
- "7b9" → "7"
- "9" → "7"
- "6" → removed
- "#5" → removed
- "maj" → "mj"

After cleanup, this was determined to be a sufficiently unique way of representing harmony for the purposes of simulated jazz improvisation.

4.18 Extension of Harmonic Symbols to Include Chord Changes

After a number of experiments using only the chord prepended to measure, results were somewhat disappointing. After consideration, a change was made based on an insight into the way that real jazz musicians approach improvisation.

An experienced musician is not only aware of the chord currently being played, but also the next chord that will be played, and improvises not just based on the chord-of-the-moment, but also based on where that chord is headed. One often hears reference to "playing the changes". For example, a dominant 7 chord resolving to a major chord will have different considerations than the same dominant 7 chord resolving to a minor chord. A solo that plays only based on the current chord will likely sound stale or sterile, and will lack a sense of direction.

Given that the network is designed to "guess" at the next upcoming character (thus, notes in the case of ABC notation) based on previously seen characters, then in order to have the same predictive ability as real musician it must have knowledge of the next upcoming chord before it has "seen" the next measure of music. With the chord symbols only prepended to each measure, the network lacks this same "awareness" of the upcoming chord.

In an attempt to fix this, the chord symbols prepended to each measure were extended to include the next upcoming chord as well - resulting in a set of "changes" prepended to the measures, instead of individual chords. The notation of the combined chord-change symbols ended up as in the following example.

A tune with this chord progression:

- C — Am — G — C

would be notated with these combined changes:

- “C”->Am
- “Am”->G
- “G”->C

4.19 Training Network on Jazz Data Set

Training for the gypsy jazz corpus proceeded in the same way as the classical guitar corpus, albeit with a shorter sequence length seeming to lend itself to better results - which is not surprising given that the gypsy jazz corpus contained almost no polyphony and so used fewer characters on average for the same length of music (polyphony requires more characters to represent).

4.20 Harmonic Context Injection During Sampling

Sampling for the gypsy jazz corpus proceeded similarly to the elaboration-targeted classical guitar experiments, with the following addition, used to simulate constrained improvisation:

A standard tune was selected to “improvise” over, with known chord changes. The set of combined chord-change pairs was pre-computed for the tune by manually typing the chord progression and using a script to process into adjacent pairs.

The sampling process was initiated by priming with the standard headers, the key of the chosen tune, and the first combined chord-change pair.

The network was then allowed to sample for a sufficient length to guarantee that it would reach the end of a measure (invariably it would “improvise” many measures of music with many “improvised” chord changes that were not necessarily the desired changes). The resulting sample was cut off at the end of the first “improvised” measure, and the next pair of combined chord-changes from the selected tune was

appended. Sampling then proceeded again, and was cut again, and the process repeated as such until the end of the tune was reached.

4.21 Converting Simulated Improvisation Results from ABC to MIDI

Conversion of gypsy jazz results back to MIDI proceeded essentially the same as the classical guitar corpus, but with the addition of a script to remove chord change symbols from the results in order to have valid ABC notation before using the *abc2midi* program to convert back to MIDI.

4.22 Listening to MIDI Results of the Gypsy Jazz Corpus

Rendering for the gypsy jazz corpus was done using the Native Instruments Kontakt classical guitar and jazz guitar sound banks, with the results aligned on top of a pre-recorded backing track for the particular tune that was improvised over.

Chapter 5

Results

5.1 Classical Guitar Data Set

5.1.1 Audio Examples

Some hand-selected audio examples:

- `http://lapompejazz.com/public/classical_guitar_sample_1.mp3`
- `http://lapompejazz.com/public/classical_guitar_sample_2.mp3`
- `http://lapompejazz.com/public/classical_guitar_sample_3.mp3`

5.1.2 Network Output Example

A truncated example of sampling results from the classical guitar data set:

```
X: 1
M: 2/4
L: 1/16
Q:1/4=70
K:F % 1 flats
V:1
%%MIDI program 24
[A/2-A,/2][A/2-C/2][A/2-F/2][A/2C/2] [c/2-A,/2][c/2-C/2][c
    /2-A,/2][c/2C/2] [A/2-F,/2][A/2-A,/2][A/2-C/2][A/2A,/2] [
    D/2-A,/2][D/2-A,/2][D/2-C/2][D/2A,/2]| [F/2-D,/2-][F/2C
    /2-D,/2-][E/2-C/2D,/2-][E/2D/2D,/2] F/2-[F/2-A,/2][F/2-B
    ,/2][F/2D/2] [D/2G,/2-][F/2G,/2-][E/2G,/2-][D/2G,/2] C/2B
    ,/2A,/2G,/2| [F/2-D,/2-][F/2-E/2D,/2-][F/2-D/2D,/2-][F/2C
    /2D,/2] B,/2A,/2G,/2F,/2 [E/2-C,/2-][E/2-C/2C,/2-][E/2-B
    ,/2C,/2-][E/2C/2C,/2-] [G/2-C,/2-][G/2C/2C,/2-][D/2C
    ,/2-][E/2C,/2]| [F/2-D,/2-][F/2-A,/2D,/2-][F/2-D/2D,/2-][
    F/2F,/2D,/2] F,/2E,/2F,/2G,/2 [A,/2-F,/2-][A,/2-G,/2F
    ,/2-][A,/2-A,/2F,/2-][B,/2A,/2F,/2-] [C/2-F,/2-][C/2F,/2E
    ,/2][D/2-F,/2-][D/2G,/2F,/2]|
[C/2-A,/2-][C/2B,/2A,/2-][C/2A,/2-][A,/2A,/2] [F/2-B,/2-][F
    /2-B,/2-A,/2][F/2-B,/2-G,/2][F/2B,/2F,/2] E,/2-[E/2E
    ,/2-][D/2E,/2-][C/2E,/2] B,/2A,/2B,/2C/2| A,/2-[A,/2-A
    ,,/2][A,/2-C,/2][A,/2A,/2] E,/2-[C/2E,/2-][F/2E,/2-][E/2E
    ,/2] D/2E/2[F/2A,/2-][E/2A,/2-] [D/2A,/2-][C/2A,/2-][D/2A
    ,/2-][F/2A,/2]| [G/2E,/2-][F/2E,/2-][G/2E,/2-][A/2E,/2-]
    [B/2-E,/2-][B/2-F/2E,/2-][B/2-E/2E,/2-][B/2D/2E,/2] [E/2-
    A,/2-][E/2-C/2A,/2-][E/2-B,/2A,/2-][E/2C/2A,/2-] [D/2-A
    ,/2-][D/2-A,/2-F,/2][D/2-A,/2-E,/2][D/2A,/2D,/2]| E,/2F
    ,/2G,/2A,/2 B,/2-[B,/2-G,/2][B,/2-F,/2][B,/2E,/2] C,/2F
    ,/2G,/2F,/2 E,/2D,/2C,/2B,,/2|

...etc
```

Figure 5.1: Classical guitar network output - ABC notation

5.1.3 Sheet Music Examples

Examples of sheet music results from the classical guitar data set:

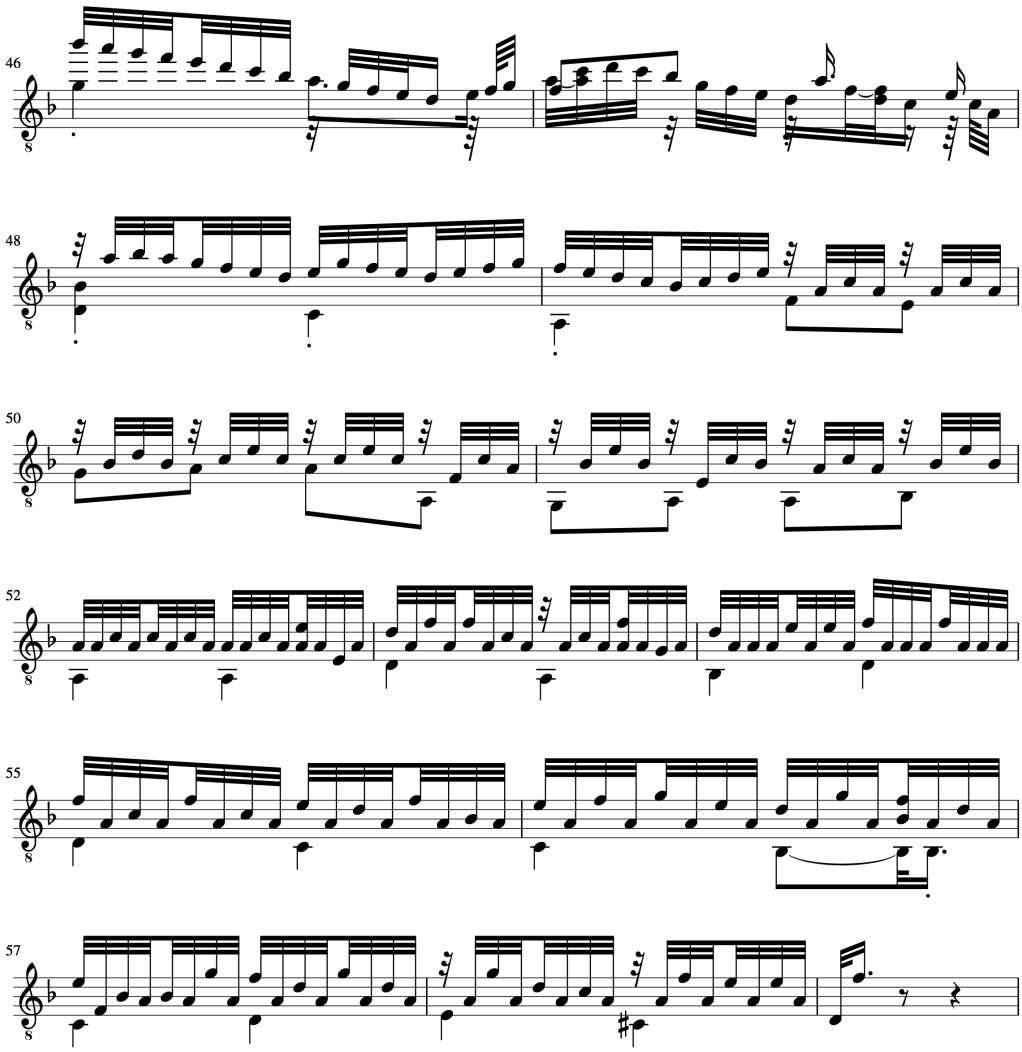


Figure 5.2: Classical guitar network output - sheet music example 1

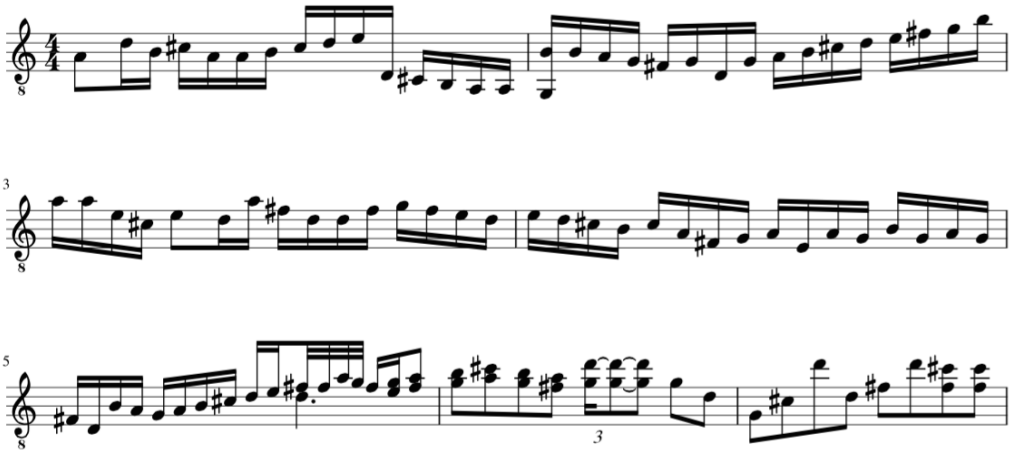


Figure 5.3: Classical guitar network output - sheet music example 2

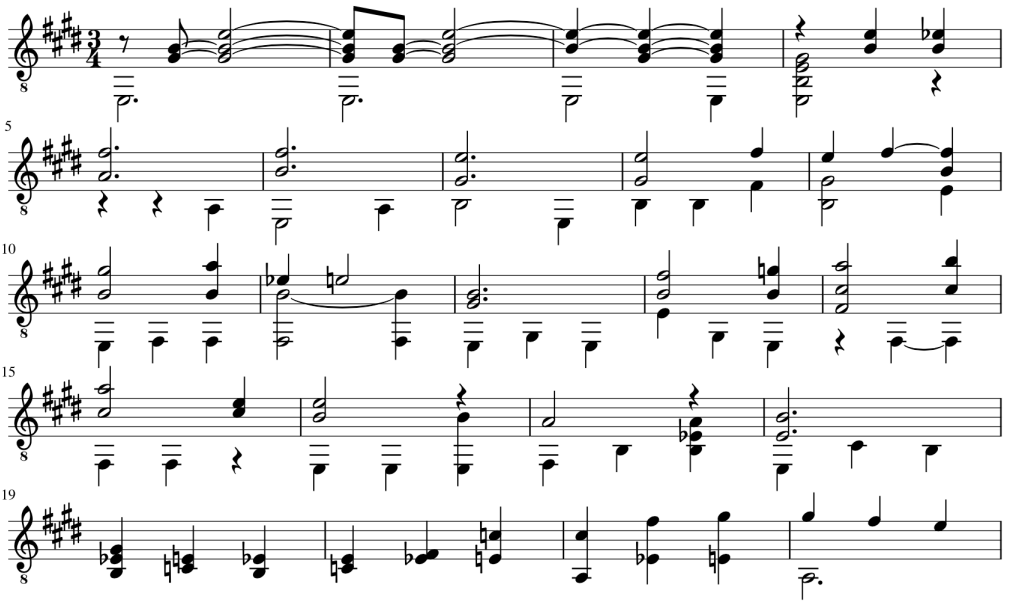


Figure 5.4: Classical guitar network output - sheet music example 3

5.2 Gypsy Jazz Data Set

5.2.1 Audio Examples

Some hand-selected audio examples:

- `https://soundcloud.com/andyhannum/djangos-tiger-june-6-2016`
- `https://soundcloud.com/andyhannum/nuages-combo`

5.2.2 Network Output Example

An example of sampling results from simulated improvisation, to the tune of Django Reinhardt's composition *Nuages*:

```
M:4/4
L:1/8
K:G
V:1
"Eb7"->D7 z4 d'bbg |
"D7"->G ed B2 b3 d'- |
"G"->G d'2 c'a g^feg |
"G"->Eb7 fe c2 B3 z |
"Eb7"->D7 z2 GA cegb |
"D7"->G d'2 ad'- d'2 d2 |
"G"->G d'2 c'=c' bggd |
"G"->F#0 {/^d} e3 f- g2 z2 |
"F#0"->B7 z4 f'c'af |
"B7"->Em dd- d4 z2 |
"Em"->Em z2 ^c'd' d'c'ba |
"Em"->Ab7 fe a2
"Em"->Ab7 ec B2 |
"Ab7"->A7 b3 a- a4 |
"A7"->Db7 z2 a2
"A7"->Db7 b4 |
"Db7"->D7 z4 a2 c'2 |
"D7"->Eb7 d'2 ba fe=dd- |
"Eb7"->D7 d4 z2 d2 |
"D7"->G ec ^c2 d4 |
"G"->G z4 d'_e'd'c'- |
"G"->Ab7 c'=b e2 e2 z2 |
"Ab7"->G7 z2 ^cd fd_dd- |
"G7"->C dc B2 c4 |
"C"->C z4 d'c'ag |
"C"->Cm ed _B2 c4 |
"Cm"->Cm z4 c'_d'c'b |
"Cm"->G {b_b}
"Cm"->G af c2 c3 z |
"G"->G z4 d'_e'd'c'- |
"G"->Eb7 c'3 b- b2 z2 |
"Eb7"->D7 z4 b2 c'2 |
"D7"->G d'2 b4 g2 |
"G"->G g4 z2 ed |
"G"->NC
```

Figure 5.5: Simulated improvisation network output - ABC notation

5.2.3 Sheet Music Example

An example of sheet music results from simulated improvisation, to the tune of Django Reinhardt's composition *Django's Tiger*:

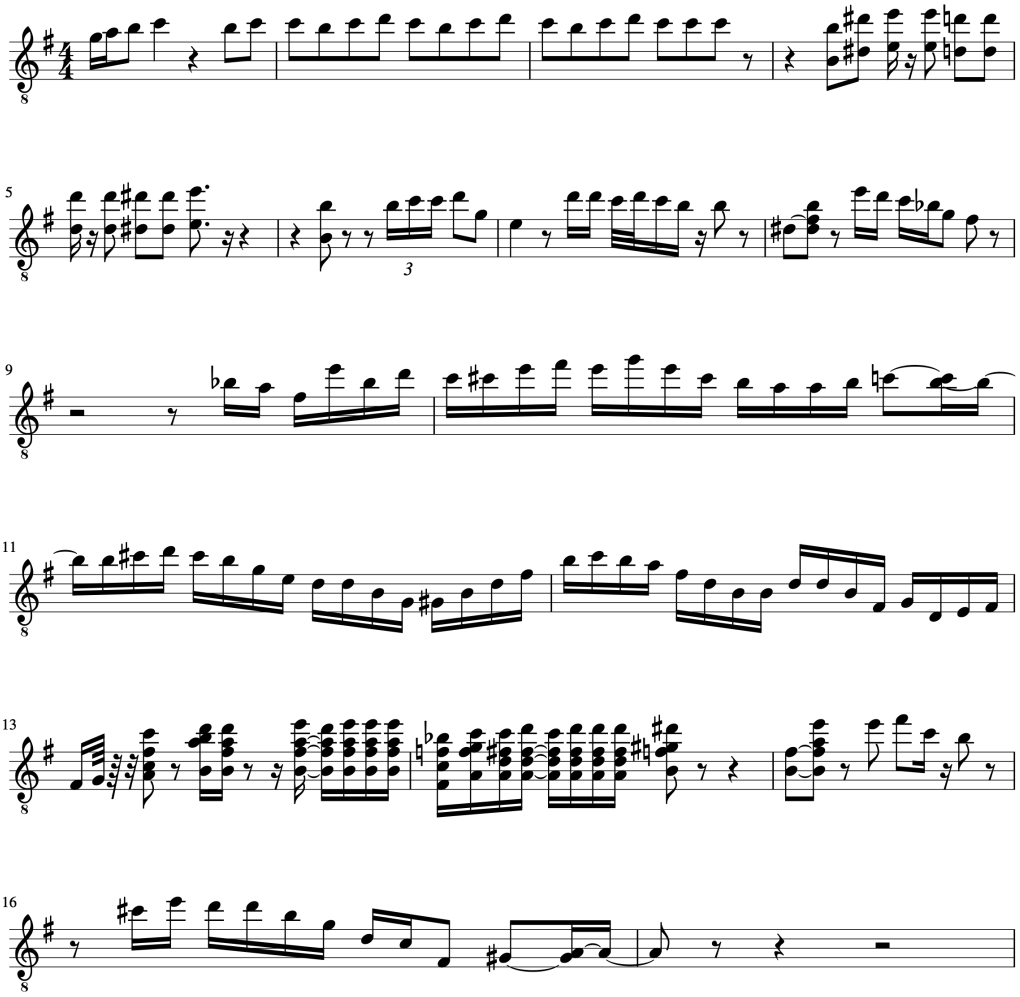


Figure 5.6: Simulated improvisation network output - sheet music

Chapter 6

Discussion

6.1 Perils of Large Data Sets and Consistency

There may be concerns with the use of large, uncurated data sets regarding the consistency and cleanliness of the data.

For example, if training on an English-language data set that included a large number of spelling errors, it may be that such errors (or even new errors due to inconsistencies) would be reproduced in the sampled output. If such errors in output were observed, it might be difficult and time-consuming to track down their source (i.e.: was it errors in the input data, or in the learning of the network?).

With music, some degree of imprecision is acceptable that might not be in other applications. If cleanliness and consistency were more important, rather than cleaning based on some heuristics it might have been necessary to do a large-scale analysis of the data set to find and fix such problems.

In this case such rigorousness seems to have been unnecessary to produce acceptable results.

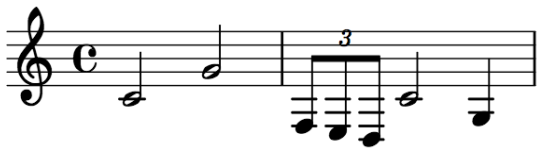
6.2 Ideal Text Format for Neural Network Learning

Despite being the format used for the original Irish Folk project, considerable deliberation happened before settling on using ABC notation for this project. In the case of the Irish Folk project, it is entirely possible that ABC notation was selected simply because of the availability of the corpus, rather than any notion that ABC was an ideal format for training the network. Other formats that were considered for this project were MusicXML and LilyPond[1]. See Figure 6.1 for examples of each.

The most significant feature in favor of ABC notation over these formats was ultimately its concision and level of abstraction. Both MusicXML and LilyPond are not simply music notation formats, but rather sheet music typesetting formats. Meaning they are intended not just to represent music, but to represent the visual layout of the sheet music. The characters used to represent the type-setting aspects are superfluous to the actual music content (which is the main target of the network training). For this reason, both of those formats use more text on average than ABC notation to represent that same amount of music. As mentioned in the methodological considerations, simplicity was preferred and non-essential characters were considered a waste of resources, so ABC notation seemed a better fit for training than the other text notation formats.

In addition, the use of ABC notation was motivated out of practicality - not because of the availability of a large corpus, but because of the availability of the necessary conversion tools.

[1] http://lilypond.org/



ABC Notation:

```
C4 G4 | (3FED c4 G2 |
```

Lilypond:

```
c'2 g'2 \times 2/3 { f8 e d } c'2 g4
```

MusicXML:

```
<note default-x="75.18" default-y="-50.00">
  <pitch>
    <step>C</step>
    <octave>4</octave>
    </pitch>
  <duration>6</duration>
  <voice>1</voice>
  <type>half</type>
  <stem>up</stem>
  </note>
<note default-x="306.14" default-y="-30.00">
  <pitch>
    <step>G</step>
    <octave>4</octave>
    </pitch>
  <duration>6</duration>
  <voice>1</voice>
  <type>half</type>
  <stem>up</stem>
  </note>
</measure>

...etc
```

Figure 6.1: Comparison of ABC Notation, Lilypond, and MusicXML

6.3 Properties of the Classical Guitar Network's Output

Evaluation of the results is primarily subjective - the main metric of quality being something akin to "musicality" of the resulting music. However, there are nevertheless still some objective evaluations that can be made of the results.

The main takeaway from the classical guitar corpus is that the network has successfully learned to reproduce polyphonic music, with little to no errors, and thus validated the main experimental premise on which the corpus was chosen. As well, the polyphony it has learned to reproduce is musically coherent and not just random.

By the metric of musicality, compared to the results of other RNN attempts, this network has performed as well as, or better than, other results. It has maintained the musical coherence achieved by the Irish Folk corpus, while making progress in complexity by way of polyphony.

6.4 Working with a Smaller Data Set for Jazz

As with large data sets, working with a smaller data set in the case of the gypsy jazz corpus came with its own concerns.

The primary concern was that there might not be enough data to train the network effectively. The credibility of that concern was confirmed in the difference between minimum validation loss seen in the classical guitar corpus training versus that seen in the gypsy jazz corpus training. The classical guitar validation loss usually bottomed out below 0.5, while the gypsy jazz corpus reached a minimum around 1.0, never going below 0.9. This result is not surprising, as it seems to be the case that a larger data set will always produce better (more generalizable) results in any machine learning approach.

What is impressive is that the gypsy jazz corpus training results nevertheless resulted in reasonably musical results. This is perhaps a result of the degree of imprecision that using a musical data set allows, especially in the genre of jazz where sometimes "outside" playing is not only acceptable but encouraged.

6.5 Choice of Chord Representation Format

Chords and chord changes were represented symbolically by a standard text format commonly found in jazz charts. This choice of representation format was not an intentional choice, but rather proceeded directly from the original source transcriptions that were obtained, and the conversion processes that followed. Although the format that resulted produced reasonably impressive results, it is possible that another choice of representation format may have benefits over the one used.

As it stands, the text format used does not reveal anything about the relationship (the harmonic similarities or dissimilarities) between different chords. For example, just based on the chord symbols alone there is nothing to indicate that a "Bm7b5" chord and a "Dm6" chord consist of entirely the same notes. However, an experienced improviser would likely be aware of that connection, and may not even necessarily distinguish between the two chords in their improvisations.

With a large enough data set, it might be expected that such similarities would be intuitively gleaned by the network by learning what types of melodies are played over such chords. But given the small size of the data set used, there may not have been enough examples to generalize the similarities based on the melodies played over the different chords. It might have been more effective to use a chord representation format that made such harmonic similarities explicit. For example, by representing chords with their constituent notes instead of by their symbols:

- Bm7b5 → [B, D, F, A]
- Dm6 → [D, F, A B]

or perhaps even a 12-note binary formatting simply indicating the presence or absence of a note in a chord:

- [A Bb B C Db D Eb E F Gb G] = [111111111111]
- Bm7b5 → [101001001000]
- Dm6 → [101001001000]

Without further experimentation it's not obvious what effect such a change would have, but given that this would inevitably reduce the number of different chord symbols seen by the network, it wouldn't be surprising if this allowed the network to make more generalizations and play more varied lines over the same chords. To play a D minor line over a F major chord, for example - something that would be unsurprising for a true jazz musician.

6.6 Properties of the Gypsy Jazz Network's Output

Subjectively, the network appears to be generating reasonably coherent improvisations, albeit in some cases not very interesting. They are generally more rhythmically uniform than might be expected in a real jazz solo - the network seems to get caught in long lines of notes with the same note length. However, such uniformity might not be completely unexpected in a beginning improvisor's solo, and comparing the network's output to the improvisations of jazz greats may be unfair at such an early stage of research.

The pieces generated by the network appear to follow the chord changes at least to some degree. At each chord, the output can usually be observed to hit one or

more of the chord tones (the most "inside" or "correct" sounding notes for a given chord).

In addition, the network appears to produce some stereotypical licks in the style of gypsy jazz. For example, there is a very stylistic formula identified by Ben Givan (**F1** from *The Music of Django Reinhardt* [5]) that appears a number of times in the results (see Figure 6.2).

A pleasing and somewhat surprising result is the appearance of chords in the output. The classical guitar corpus proved that homophony and polyphony could be learned and reproduced, but it was a corpus full of such examples, whereas the gypsy jazz corpus very rarely features chords. Despite that, the network occasionally produces chordal pieces of solos at seemingly appropriate places.

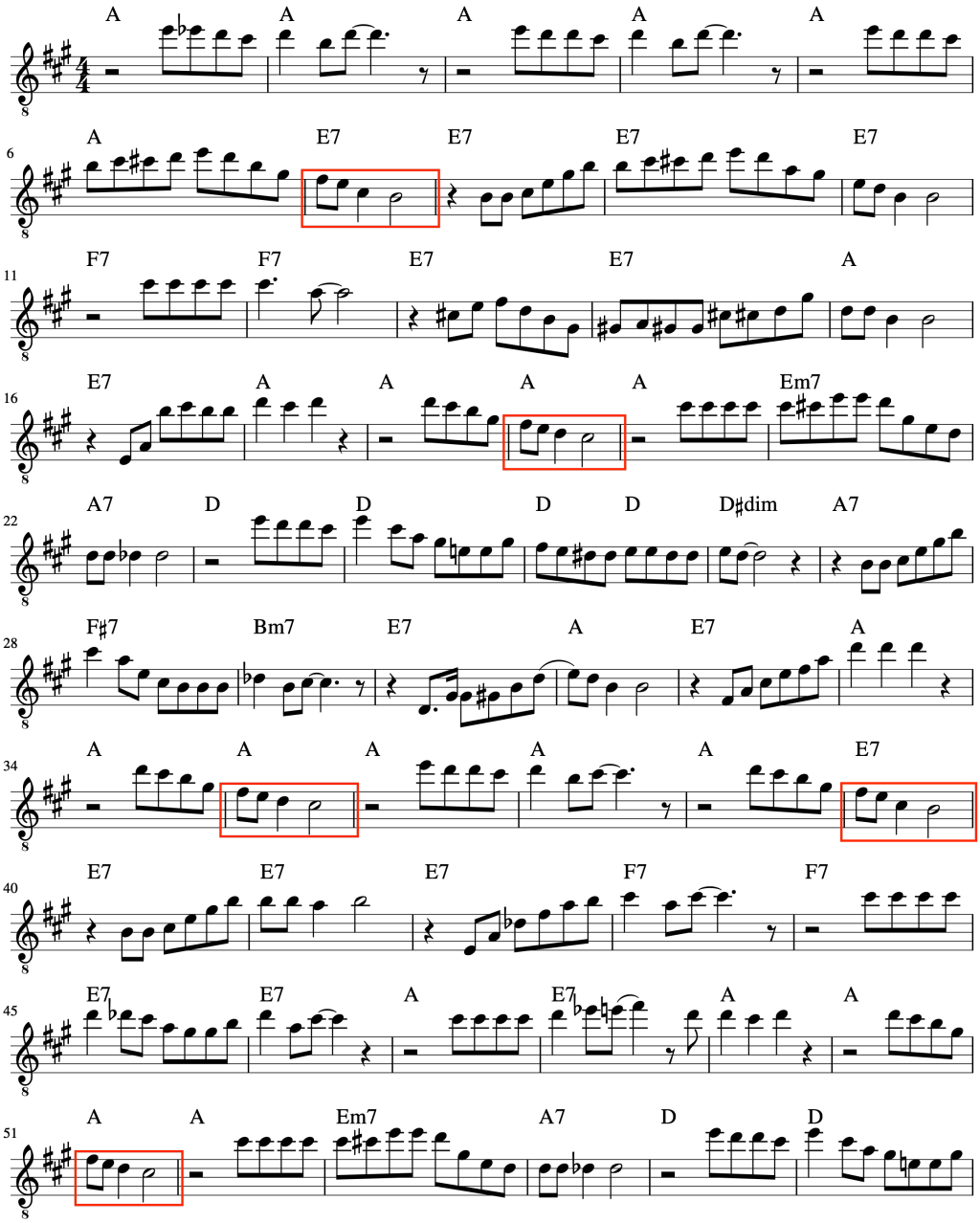


Figure 6.2: Highlighted instances of Ben Givan's formula F1

Chapter 7

Applications

7.1 Generating Musical Exercises Using the Trained Network

Given that the results of the algorithmic generation process were found to be generally musical-sounding, the next step was to attempt to use the trained network to generate exercises aimed at musical pedagogy.

Development of such exercises is not as simple as producing musical-sounding content, it is also a matter of controlling the parameters of the generated music in a way that is effective for teaching. The final use of such parameterization would be best determined by a teacher (say, a classical guitar teacher) so the techniques developed here were aimed at giving such an end-user as many options as possible, to use at their discretion.

There are multiple ways this parameterization could have been approached, with perhaps the most obvious being to attempt to control the network's output during the sampling stage to produce the desired properties. Such an approach bears further investigation, however the technique chosen here was to make no changes

to the output of the network, and instead develop a set of post-processing tools to achieve this end.

7.2 Post-Processing Tools for Exercise Parameterization

A web application was developed as an all-in-one tool to maintain a database of potential exercises, to allow an end-user to filter and select appropriate exercises based on a variety of parameters, and ultimately to continue to grow the database of exercises based on user feedback.

The network is allowed to generate large amounts of non-constrained musical ideas, all of which are then split into equal-length musical phrases (8 measures by default).

Each piece, now considered a "potential" musical exercise, is run through a custom Ruby script that analyzes the MIDI for a set of filtering properties, and through a custom MuseScore script that both renders the MIDI to sheet music in PNG format, and analyzes the resulting sheet music for an additional set of filtering properties.

The full set of properties analyzed and saved for each potential exercise is as follows:

time_signature The time signature of the piece.

musescore_shortest_duration The shortest note duration.

musescore_longest_duration The longest note duration.

musescore_longest_repeat The length of the longest chain of repetition of the same note or chord in a row.

musescore_num_elements The total number of visual elements in the sheet music rendering.

musescore_num_durations The total number of different note durations.

musescore_num_chords The total number of chords.

musescore_num_notes The total number of notes.

musescore_num_ties The total number of ties.

musescore_num_rests The total number of rests.

musescore_num_sharp_flat The total number of accidentals not already in the key signature.

musescore_key_signature The key signature in number of flats (-) or sharps (+).

entropy_pitch_class The entropy[1] calculated from all pitch names.

entropy_interval The entropy calculated from all intervals between adjacent notes.

entropy_rhythm The entropy calculated from all note durations.

low_note The pitch value of the lowest note.

high_note The pitch value of the highest note.

interval_range The difference between pitch value of the lowest note and the highest note.

The web application stores these exercises along with their properties in a SQLite database, and displays a simple interface to the end-user allowing them to directly view and listen to exercises, and successively filter the exercises based on ranges for any of the above parameters.

[1] $H(X) = -\sum P(x_i) \log_b P(x_i)$

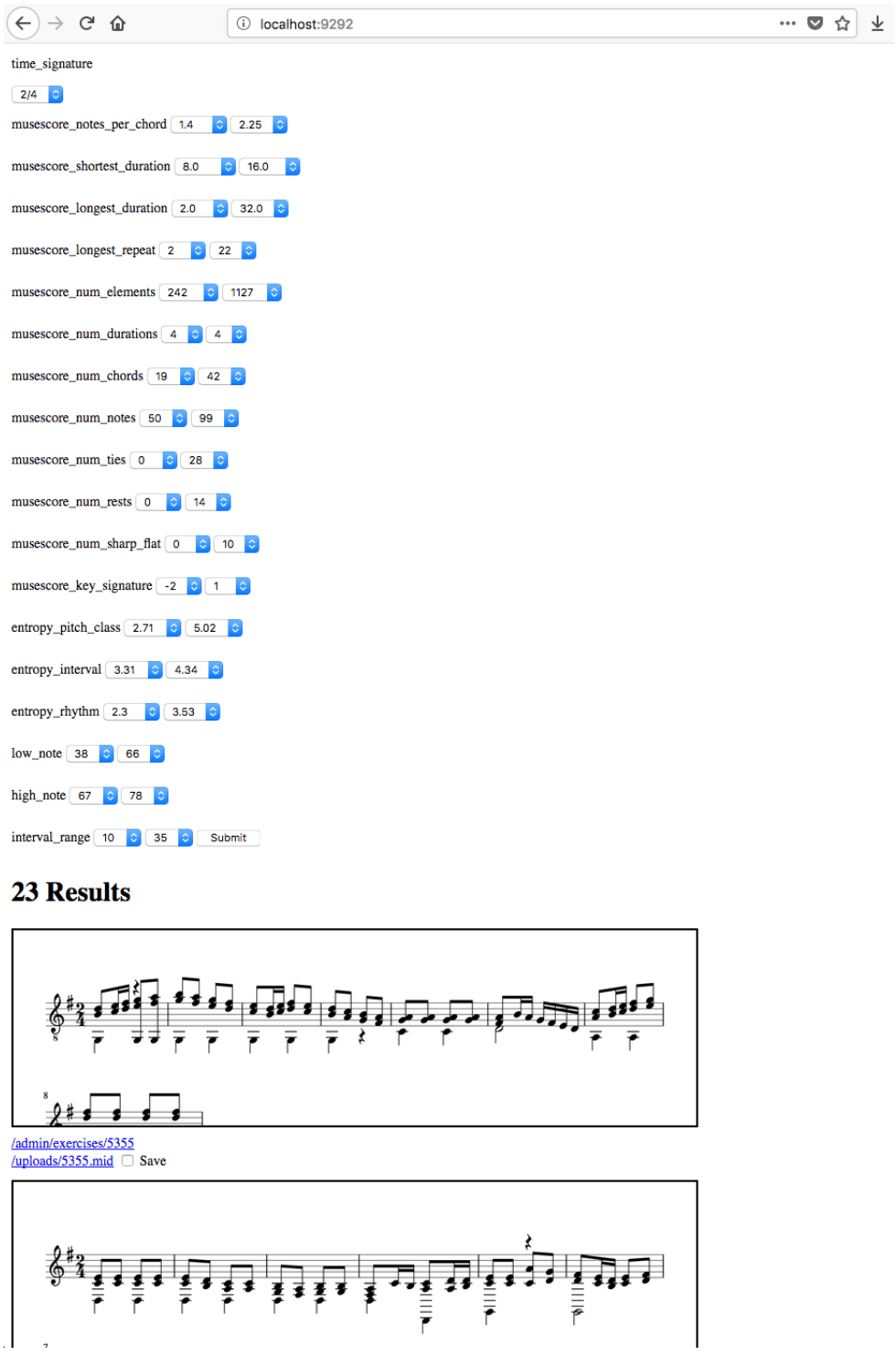


Figure 7.1: Exercise filtering web application

7.3 Growing Exercises via Human Feedback and Evolutionary Algorithm

After finding exercises that fit some criteria, the end-user can save selected exercises to be used for "growing" the database.

The selected exercises are re-introduced to the trained network and elaborated upon using the technique previously detailed in the methods. The results of the elaborations are then split into exercises and re-processed back into the application to be added back into the database of exercises.

The intention is that such elaborations will be similar to, but distinct from, the exercises that were selected by the user to fit their criteria. So, by repeating this process of selection and elaboration the user will be able to "grow" a database of exercises that fit their specific needs from potentially only a few original example exercises.

Ultimately, if a hand-selected database was grown large enough, the resulting exercises could be used to further train the network so that the results of the immediate generation step are more in line with the desired output. With enough examples it could be feasible to train a network that exclusively generates exercises with the desired parameters, and requires no post-processing filtering.

This labeling and evolutionary strategy could also potentially be used to grow databases of musical snippets which conform to other hand-selected characteristics that are not necessarily quantitative. For example, a human could label pieces of music with their emotional mood, and eventually grow a database of music snippets that match specific moods.

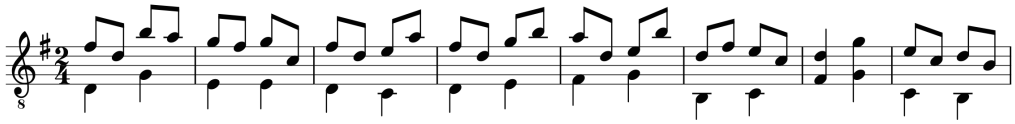


Figure 7.2: Example of original exercise

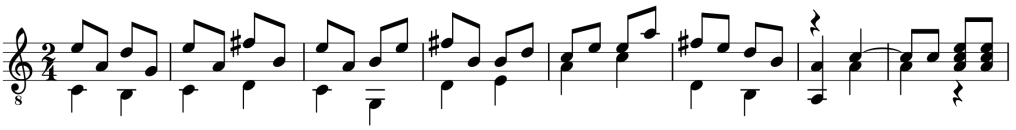


Figure 7.3a: Exercise elaboration 1

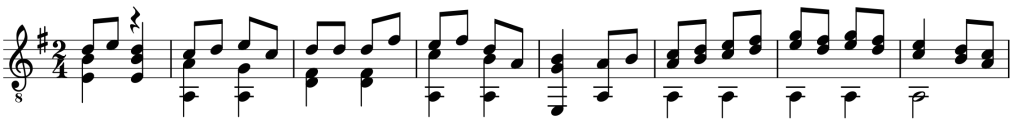


Figure 7.3b: Exercise elaboration 2

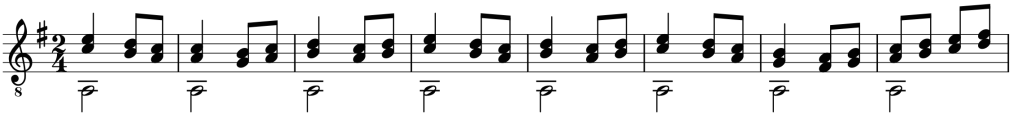


Figure 7.3c: Exercise elaboration 3

Chapter 8

Future Work

8.1 Obtaining Larger and More Specialized Data Sets

The main limiting factor of these experiments, and the impetus of the project as a whole, was the limited availability of appropriate data sets for training. Despite the subjective quality of the results, there remain a number of reasons to continue attempts to procure more input data. The most obvious being that more data simply provides more examples for the network to learn from, which should theoretically produce better results if the additional examples exhibit the qualities that we would like the network to reproduce.

More to the point, the size of the data set used in the classical guitar experiments was small enough to warrant artificial inflation (by way of transposing to different keys) and the size of the data set used in the gypsy jazz experiments was small enough that the addition of less than 25% more examples noticeably improved results. Both data sets seem ripe for more input examples.

In addition to increasing the amount of data overall, it would be interesting to be able to run experiments that were more focused stylistically. In the case of the classical guitar experiments, pieces from all composers were combined in the

training input, despite many of the composers having drastically different styles, and indeed having composed during drastically different time periods. If a larger corpus of pieces were available from a single composer, or from stylistically similar composers, then experiments could be run with better expectations about the style of the output than simple subjective musicality. Output might be expected to display certain predefined characteristics of the input composers' styles.

Such stylistically-focused experiments may be more useful to the production of pedagogical exercises, as the output would be expected to exhibit more regularity and be more formulaic in general (due to higher expected regularity in the input) than output of experiments where the input was an amalgamation of composers of different styles.

This approach can somewhat be seen in the gypsy jazz experiments (where some stylistic Django formulas were seen in the output), though even there the final experiments used a combination of pieces from different gypsy jazz performers due to lack of sufficient examples of a single composer.

Searching for new data sources remains an ongoing process. Two potential sources that might be worth investigating are commercial sheet music and transcription services (i.e. buying the digitized sheet music), or crowd-sourcing (i.e. asking the public at large for donations of their transcriptions).

8.2 Controlling Different Aspects of Complexity Independently

Despite the tooling developed for the post-processing style of exercise generation, there remains a strong appeal regarding the ability to control different aspect of music complexity at the network level - during the actual sampling stage, rather

than afterwards. Not only would such a solution likely be more efficient, but it would also be more elegant.

The sampling *temperature* parameter in some ways provides a loose approximation at this kind of control. Sampling with significantly higher temperature will almost always produce more complex results, whereas sampling with significantly lower temperature will almost always produce simpler results.

However, complexity is a multidimensional aspect of music, and as such temperature is a poor approximation of a true musical complexity "knob" for a number of reasons. Foremost is that it does not distinguish between different dimensions of complexity - for example it cannot distinguish between a piece that is rhythmically complex but melodically simple, or melodically complex but rhythmically simple. Furthermore, changes to the temperature parameter also often negatively affect the degree of musicality of the output (both more complex and less complex output is often less musical; there appears to be a sweet spot in temperature for musicality) and also the coherence of the output (higher temperature results in more errors in the ABC notation produced).

In theory it seems that the ideal setup would be to have additional inputs into the network that parameterize different aspects of the musical complexity (rhythmic vs melodic complexity for example). During training, pieces presented to the network could be analyzed for their relative complexities on these scales, and the resulting measurements could be added to the input layer. Then after training, during sampling, these additional inputs should theoretically be able to provide control of the complexity of the output on independent axes. One difficulty in such an implementation would be developing a useful way of measuring "complexity" in the different domains.

This is just one of many possible approaches that could be explored in the future to provide a better model for directly generating musical exercises.

8.3 Learning Larger-Scale Musical Structures

As seen through the artistic/entertainment lens, a significant weakness in the output from the network is that, although the music can be observed to exhibit coherence in local structures (within measures and between adjacent measures), it fails to exhibit any sort of "understanding" of large scale structures that would be necessary for composing fully-formed songs.

If the ultimate goal is fully algorithmic music composition (as opposed to exercise generation, for example) then solving the problem of creating larger scale structures might be a good next step.

Attempting to solve this problem using a neural network like *char-rnn* seems tempting, and indeed there have been attempts to do something like this [9]. A potential future project might try to apply such techniques to the classical guitar data set.

Another approach, beyond modifying the neural network architecture, and hoping that it will learn via examples, would be to algorithmically generate large-scale song structure via other, more traditional means, and then once the structure has been decided, use a network such as this one to "fill in the details" of local musical ideas. The results of this project have shown that structure can be effectively forced on the output of this particular network during the sampling stage by the use of priming.

Results of a somewhat similar approach using LSTMs to generate music given a pre-defined rhythmic structure were published in 2016, with audio examples being quite pleasing to listen to [11].

Bibliography

[1] Bruce Benward. *Music in Theory and Practice*. McGraw-Hill, 1977.

[2] David Cope. *Experiments in Musical Intelligence*. A-R Editions, 1996.

[3] Ryan Davis. Algorithmic Music Generation for Pedagogy of Sight Reading. Master's thesis, University of Denver, 2018.

[4] Douglas Eck. Finding Temporal Structures in Music: Blues Improvisation with LSTM Recurrent Networks. *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, 2002.

[5] Benjamin Givan. *The Music of Django Reinhardt*. University of Michigan Press, 2009.

[6] Mohamad Hassoun. *Fundamentals of Artificial Neural Networks*. A Bradford Book, 2003.

[7] Daniel Johnson. Composing Music With Recurrent Neural Networks. http://www.hexahedria.com/, 2015.

[8] Ji-Sung Kim. DeepJazz. https://deepjazz.io/, 2016.

[9] Adam Roberts. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. *ICML 2018*, 2018.

[10] Peter M. Todd. A Connectionist Approach to Algorithmic Composition. *Computer Music Journal, Vol. 13, No. 4*, 2008.

[11] Christian Walder. Modelling Symbolic Music: Beyond the Piano Roll. *JMLR: Workshop and Conference Proceedings 63:174–189*, 2016.