­

Jazz Piano Lead Sheet Arrangement via deep learning

Elliot Hogg

200817750

David Herbert

Mark Turner

Jim Mcgrath

DD/mm/yyyy

15000 Words

Declaration

I hereby declare that this dissertation represents my own work except where otherwise stated.

Abstract

Over the past 10 years, deep learning has been extensively researched and used in the task of music generation. However, one area that remains mostly untouched, is the task of lead sheet arrangement. In the genre of Jazz, most music is represented in the lead sheet format – a single notated melody and chord symbols. The interpretation and arrangement of the music must be decided and undertaken by the performer. To tackle this problem, a novel conditional generative adversarial network (C-GAN) was designed to generate jazz chords which could be used to create full arrangements. Participant lead evaluations showed that resulting arrangements were found to be more effective than template chord selection.

Acknowledgments

I would firstly like to thank my supervisor David Hebert, whose support and advice was invaluable to this project.

I would also like to thank Mark Turner for his continued help and support throughout the project.

Contents

1. Introduction (1700 words)

2. Background Research (1700 words)

(6600 words):

4. Implementation

4.1. XML Chord Scraper

4.2. Deep Learning Models

4.3. Lead Sheet Arranger

4.4. Complete System ??

(3300 words):

5. Results and Evaluation

5.1 Scraper Accuracy

5.2. Deep Learning

5.3. Arrangements

5.4. complete system

5.3.1. Participant survey

5.3.2. Technical evaluation

(1700 words):

6. Conclusion

6.1. Meeting aims and objectives

6.2. Effectiveness of approach and systems

6.3. Contribution to the field

6.4. Future Considerations

1. Introduction

Lead sheet arrangement is a task whereby a musician takes an abbreviated representation of a song and adds harmonic and rhymical elements to transform it into a compelling piece of music. In the area of jazz music, the lead sheet is the traditional and most common way for songs and pieces of music to be notated, mainly due to the fact that they allow for creative freedom and leave much to the interpretation and direction of the musician, an idea which is essential to jazz music. Figure 1 shows the first 4 bars of a piano lead sheet.

Lead sheet arrangement can be divided into two subtasks: (1) Harmonic – choosing a set of notes to be played for each chord symbol. This task is also referred to as *chord voicing*. (2) Rhythmic – temporally arranging chosen chord notes to form a rhymical pattern. Figure 2 below highlights these subtasks.

A black and white photo of a piano

Description automatically generated with low confidence

Figure 1. The first 4 bars of the lead sheet for the song Over the Rainbow by Errol Garner. The melody is represented as western notation, and the chord types are indicated by the chord symbols above.

Diagram, engineering drawing

Description automatically generated

Figure 2. Lead sheet arrangement of the first bar from figure 1. From left to right, a C chord symbol (shown in red) is first represented harmonically, and then arranged rhythmically.

Computationally, lead sheet arrangement can be defined as a process that takes a lead sheet as input, and outputs a harmonically and rhythmically complete digital representation of that song.

There are a few commercial applications, such as *Band in a box* and *iRealPro,* that perform the harmonic element of lead sheet arrangement, however they are reliant on simple selection algorithms and a limited dictionary of chord voicings. The chord voicings themselves are generic and lack musicality. There are however no applications that attempt to perform the rhythmic subtask of lead sheet arrangement, arguably due to this being a much more complex task (citation).

There has been some academic research of traditional programming approaches to the harmonic task of lead sheet arrangement that have shown to outperform the aforementioned applications, however the results of traditional programming approaches lack variation, as they struggle to output multiple different sets of notes for the same chord symbol (citation). They also rely on being preprogramed with a set of musical theory rules.

The use of deep learning in lead sheet arrangement has seen very little academic research. There exists only two publications on the topic, both of which have the same authors. There has also been some recent publications using deep learning in the task of chord voicing generation, which is essentially the same task as the harmonic arrangement task in lead sheet generation. These papers are critically evaluated in section 2.x.

Although there is limited research on the task of music arrangement, the task of musical generation has seen a large amount of attention (citation). This literature is valuable to this project, as arrangement can be defined as a subtask of generation. Surveying an evaluating recent music generation research can greatly contribute to the direction of this project. Especially in regard to gaining an insight into which deep learning models would be most effective for this project. Section 2.x surveys music generation literature in search of effective deep learning algorithms.

And the focus on music composition within the field of deep learning has

has regained a large amount of academic and public attention within the past 10 years (citation). This is primarily due to the development of deep neural networks (Ji et al., 2020: 5). One great example of such generation is *DeepBach*, a deep learning model that can compose novel, convincing chorales in the style of composer Bach (Hadjeres et al., 2017). Deep learning has also been used to generate monophonic and polyphonic music of many different genres (citations needed).

Research and implementation of deep learning in the task of lead sheet arrangement is however very limited. There exist only two publications on the topic, both from the same authors. In their research, they employed deep learning to arrange pop lead sheets into a full arrangement for a 4 piece band (reference).

Music generation tasks are usually achieved using either recurrent neural networks (RNN) or generative adversarial networks (GAN). The former type of network is highly effective at generating temporal sequences, as the networks outputs are dependent on previous outputs (citation). The latter network is effective at generating completely new instances of data that are undistinguishable from a given set of data (citation).

The inherent nature of the lead sheet is that, once a musician has composed an arrangement, they tend to memorise it rather than notate it (citation). This is a particular issue in jazz, in which there exist very few arrangements of jazz standards in relation to the size of the total corpus (citation). For intermediate or amateur jazz musicians, or for advanced classically trained musicians, the task of lead sheet arrangement can be challenging, and can create an effective barrier to entry for the genre as a whole (reference).

Harnessing the power of deep learning in order to create a system capable of generating compelling arrangements would be invaluable to the jazz community, as it would make jazz music much more accessible.

The focus of this project is to create a lead sheet arrangement system that will take a lead sheet as input and output an arrangement. The system will also be intended to provide a framework for others to employ and use in order to advance the field of research. The system will be limited in that it will only address the harmonic element of the process, ie. It will provide chord voicings for the chord symbols.

The system will consist of three key elements. (1) A chord scraper that is capable of extracting chord voicings from fully arranged songs with chord symbols. This scraper will be limited to songs in MusicXML format, which is a popular XML based music encoding language (reference). (2) A chord generator that will use a conditional generative adversarial network to generate musically pleasing and convincing chord voicings. The network will be trained using a novel dataset of labelled chords gathered by using the chord scraper on existing jazz piano arrangements. (3) A lead sheet arranger, which will take the melody from the lead sheet and the chords from the generator, combine them, and output an arrangement in MusicXML, MIDI, and Audio formats.

The project will also present the novel dataset of labelled chord voicings for use in further deep learning tasks.

**1.2 Aim**

To create a deep learning led system that can generate arrangements of any given jazz lead sheet.

**1.3 Objectives**

* To develop a program that can scrape chord voicings from fully arranged pieces of piano music with chord symbols in MusicXML format
* To present a novel dataset of labelled jazz piano chord voicings
* To create a novel conditional generative adversarial network that can output compelling colourings and voicings for chord symbols from a given lead sheet
* To develop a program that can combine chord voicings with the melody of a lead sheet in a way that is both convincing and playable.
* To perform some empirical and participant led testing on the outcomes.

**1.4 Outline**

The structure of the report follows both the order it was developed in as well as the order in which the system as a whole operates.

* **Chapter 2** provides a review of the recent literature within the field of deep learning as it pertains to music. There will also be a critical evaluation of existing lead sheet arrangement models.
* **Chapter 3** provides information about the data used in the project.
* **Chapter 4** provides a deep dive into how the three key components of the system were conceptualised and developed. Starting with the chord scraper, then the C-GAN deep learning model, and finally the lead sheet arranger.
* **Chapter 5** presents an evaluation of each component, as well as an evaluation of the results of the system as a whole. Evaluations of the system are both technical and participant led.
* **Chapter 6** presents a conclusion of the project, with commentary on the how initial aims and objectives were met, as well as some future considerations and recommendations for further research.

2. Background Research

2.1 Motivations

The initial motivations for this project were to make a meaningful and novel contribution to the jazz piano community within the context of a computer science project. As a keen jazz pianist with a classically trained background, the task of lead sheet arrangement has always been a challenge. In order to make an arrangement sound good, a large amount of knowledge and experience in voicing chords is required. A system that could generate chord voicings that were indistinguishable from the voicings of renowned pianists such as Bud Powell, Bill Evans, or Herbie Hancock would remove the barrier for entry to a large number of aspiring jazz pianists.

This is the motivation that led me to pursuing a research project in the field of deep learning and lead sheet arrangement. I hope that this research can at the very least, serve as a springboard for future research to be carried out in this area.

2.2 Searching for an effective deep learning model

As mentioned in the introduction, there is a limited amount of previous research on the task of lead sheet arrangement using deep learning. However, as arrangement is a subtask of generation, research on music generation is also highly relevant to this project. In this subsection, lead sheet arrangement research and music generation literature will be surveyed in order to assess the capabilities of current research and see how effective their models are in the task of arrangement. In doing this research, the aim is to both find pre-existing models that can be adapted to fit the needs of this project, as well as use the designs and results of existing models to guide the design within this project of a partially novel deep learning that can accomplish the task of lead sheet arrangement.

In order to thoroughly survey the landscape of arrangement and generation, a full scrape of the field was conducted, which were then filtered and reduced into a list of 35 research papers. Meta information was manually extracted from each paper, such as the deep learning models used, datasets used, and whether the models were publicly available. This table can be found at APPENDIX X.

The first paper of note was titled ‘Lead Sheet Generation and Arrangement by Conditional Generative Adversarial Network’. In this project, the authors were able to generate lead sheets, and then arrange them both harmonically and rhythmically. As the title suggests, the model used was a conditional generative adversarial network or C-GAN. The results of the paper demonstrated that this model is effective in the task of harmonic arrangement, as it was able to generate convincing chord voicings when given chord symbols.

One major limitation of this model however was that it could only generate simple chords, i.e., major or minor. It was not capable of more complex jazz chords, due to the fact that it was trained on pop songs.

Despite the limitations of this research, the results were promising. The model was available on GitHub, however, attempts to get it running failed as the list of dependencies was incomplete, and some versions of those dependencies were unavailable. Contact was attempted with the authors; however, no response was received.

This paper was very useful to this project, as it brought to our attention a deep learning model that was capable of the task of lead sheet arrangement.

A further 3 papers were evaluated in which the authors used either generative adversarial networks (GAN’s) or C-GAN’s in the task of music generation (Dong et al., 2017; Yang et al., 2017, Liu et al., 2018). Each model was able to generate novel instances of music that were indistinguishable from the training dataset.

The second paper of note was titled ‘Chord Jazzification: Learning Jazz Interpretations of Chord Symbols’. Although this paper made no mention of the lead sheet, the model presented was able to take as input a sequence of chord symbols and output a series of chord voicings to represent those symbols, which is fundamentally the same task as harmonic lead sheet arrangement.

The main insight gained from this paper was the way in which the chord symbols and chord notes were encoded. The authors presented an effective way on encoding chord symbols as numerical labels. This approach provided a useful framework for which this research project could both gather and represent its own data, of which the details are presented in section 3, 4.1 and 4.2.

The model used to generate the chords sequences were variations of recurrent neural networks (RNN’s). The reason this network was used is that it excels at generating sequential data in which each individual data instance is dependent on the instance generated before it. This type of model would be highly effective in this project aim of harmonic lead sheet arrangement, as it both generates individual chord voicings as well as ensures that they are in some way meaningfully connected as a sequence.

The one major limitation of the network was that the chord voicing sequences it generated were limited to a maximum of 8 bars in length, which is far less than a jazz piano lead sheet. The source code for this model was available on GitHub, and was downloaded and in order to investigate further.

After attempting to increase the length of the generated sequences, it was discovered that the maximum length of generated sequences was limited to the length of the sequences used to train the model. Due to a lack of data, the authors were forced to split their data into training instances of 8 bar long sequences.

This discovery suggested that RNN’s would not be suitable to the requirements of this project, as the sheer amount of data required to train them in order to output full length lead sheet arrangements would be unfeasible.

A further 7 research papers using RNN’s in the task of music generation were evaluated, confirming that the length of sequence generation was limited to the length of training sequences (Hadjeres et al., 2017; Liang et al., 2019; Mogren, 2016; Szelogowski 2021; Teng et al., 2017; Zhao et al., 2020; Zhu et al., 2020;). All of these papers can be found at APPENDIX.X.

Two further generative deep learning algorithms were highlighted in a journal titled “A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions” (Ji et al., 2020). These two algorithms were Autoencoders (AE’s) and Variational Autoencoders (VAE’s). Although these algorithms showed some good results in the task of music generation, the results of three recent papers suggest that VAE’s are not as effective as GAN’s, mainly due to the fact that they do not appear to be able to learn the true posterior distribution of a given dataset (Angioloni et al., 2020; Engel et al., 2017; Valenti et al., 2020).

In order to learn more about C-GAN’s and their use in generation tasks, a wider scope of literature was searched. There were two papers of note, which both used C-GAN’s to perform tasks that were fundamentally the same as harmonic lead sheet generation.

The first paper presented the *Pix2Pix: image translation model,* which uses a C-GAN to perform a variety of image translation tasks. For example, in one instance, the model is trained to take as input a satellite image and output a google maps style image representation. Although the data type and shape differ to that of chord symbols and notes, the translation from one representation to another applies to this project, and thus the model is highly informative and applicable.

The second paper presented a one-dimensional adaptation of the *Pix2Pix* model. This is also of significant interest to this research project, as both chord labels and chord notes are best represented as one dimensional vectors. Although this adapted model was not openly available, the author was contacted and enthusiastically agreed to share

Both of the above models are used within this research project, as seen in section 4.2.

2.3. Generative Adversarial Networks (GAN’s)

Generative adversarial networks are a type of neural network developed by Ian Goodfellow and his colleagues in 2014 (Goodfellow et al., 2014). For the reader to understand this paper, a high-level explanation of GAN’s will be provided. The purpose of a GAN is to be able to generative new instances of data from a given data distribution. For example, if a GAN was trained on a dataset of images of faces, it would then be able to generative completely new images that would exist in a similar distribution space to the training dataset.

A GAN is made up of 2 independently functioning neural networks, a discriminator, and a generator. (Goodfellow et al., 2014: 3).

The generator is a neural network that takes a noise distribution as input, and outputs a fake instance of data. To use the previous example, the generator would output fake images of faces. Initially, the fake images would be very poor, however through training the generator, it would eventually be able to output fake images that were convincing enough to be indistinguishable from images in the dataset.

The discriminator is a neural network that operates as a classification network. It takes as input both real instances of data, and generated, fake instances of data, and outputs a number indicating whether the input was real or fake.

When initially training the GAN model, the discriminator easily classifies the real instances as real, and the generated instances as fake. However, throughout training, the generator improves to produce more and more convincing instances, to the point where the discriminator is unable to effectively classify real instances as real, and fake instances as fake. At this point, the generator can produce completely novel instances of data from the given dataset distribution.

2.4. Conditional Generative Adversarial Networks (C-GAN’s)

Conditional generative adversarial networks allow the output of GAN’s to be conditioned on a given input (citation). Extending the example from 2.2 to having a labelled dataset of faces made up of faces of men, woman, girls, and boys. When training the generator, instead of passing in a noise distribution, the labels are passed in. In doing so, the generator is conditioned to effectively learn the distributions of each of the four labels. Thus, when generating faces using the trained network, one of the four labels could be passed into the generator, resulting in a generated face of that labels distribution.

2.5. Survey of available datasets

Deep learning generation models require a large amount of data for training. C-GAN’s require data that is labelled. For this research project, the data must consist of pairs of chord symbols with their respective chord voicings, i.e. the set of notes that represent a chord symbol. For more details on the data requirements, see section 3 and 4.1.

In other deep learning fields, such as image generation, extended research over the last decade has left a great number of refined corpuses for which researchers can use in their models. However, as the field of deep learning in music is still relatively small, the availability of datasets is not comparable. In order to determine if there was a pre-existing dataset that could be used or adapted to fit the model of this project, a full survey of available datasets was conducted. The survey found 15 jazz related datasets. However, after conducting a survey, a much more comprehensive list of datasets was found in Ji et al’s., 2020 deep learning in music review paper (Ji et al., 2020:56-58). Therefore, only the datasets that are not present in Ji et al.’s paper will be presented below:

|  |  |  |
| --- | --- | --- |
| Format | Type | Contains |
| Text | Jazz Audio-Aligned Harmony (JAAH) Dataset | Meter, structure, and chords of 113 Jazz tracks |
|  | Weimar Jazz Database (WJazzD) | Transcriptions of 135 jazz solos |
|  | JazzCorpus | Annotated chord progressions for 77 jazz pieces |
|  | Chord-Jazzification dataset | Annotated chord voicings from 50 pop-jazz piano solos |
| MIDI | LMD-matched | 45,129 songs matched to the Million Song Dataset |
|  | Big\_Data\_Set: “The Largest MIDI Collection on the Internet” | 130,000 songs including jazz solos |
| MusicXML | Charlie Parker's Omnibook data | Transcriptions of 50 Charlie Parker improvisations |

Table 1. A list of jazz related datasets that are available and open to use. This is an extension of Ji et al.’s 2020 dataset survey.

Following a survey of all the available datasets, 2 were identified as being of interest to this paper. (1) The Wikifonia dataset, which is a collection of 7000 jazz lead sheets. This dataset can be used in order to test the project. (2) The Chord-Jazzification dataset (see table 1). This is the only existing dataset that contains chords presented as pairs of chord symbols and chord voicings. The only limitation is that most of the chords are taken from pop chords, and do not contain the more complex chords used in jazz.

Following this survey, it was determined that a novel dataset would need to be gathered in order to train the deep learning model. More information of this can be found in section 3 and 4.1.

3. Gathering Dataset

3.1 Initial Research

Training conditional adversarial networks (C-GAN’s) require a large amount of labelled data (see 2.3 and 2.4). In the context of this project, the data is chord voicings, and the labels are their associated chord symbols. Figure 3 shows how these data pairs would be represented inside a jazz piano fully arranged score (glossary).

Diagram

Description automatically generated with medium confidence

Figure 3. Showing two data pairs, with chord symbols highlighted in red, and chord voicings highlighted in blue. The melody notes have been omitted from this score.

A review of available datasets (section 2.5) found one corpus which contained chord symbols and chord voicings; however, the chords were taken from pop songs and thus didn’t meet the requirements for this project (reference to pop vs jazz chords?).

In order to gather a novel dataset of labelled chord voicings, a large set of jazz piano solos would need to be scraped and the chords extracted. The solos would also need to be of a certain degree of quality, in that they would need to contain chord symbols that were accurate and chord voicings that were compelling.

Music is conventionally represented in either MIDI or an XML type format such as MusicXML. As MIDI does not allow for chord symbol encoding (reference), a library of jazz piano solos in XML format was required.

Looking back at the list of available datasets (2.5), there was no such data library. A further survey was conducted in which 23 XML libraries were discovered (table from <https://www.musicxml.com/music-in-musicxml/> in Appendix). The survey showed that a website called *MuseScore.com* had over 1 million scores in XML format, significantly more than any of the other libraries.

The search function on *Musescore* showed a total of 4,837 available jazz piano solos. However only 300 of the solos were fully arranged and contained chord symbols. The rest were either lead sheets, or full arrangements without chord symbols. Of the 300 suitable solos, 171 were selected. Further details of this are found in the following section.

3.2 The Dataset

171 fully arranged jazz piano solos with chord symbols were manually downloaded from *musescore.com*. In order for a solo to be selected, it must have met all of the following criteria:

1. Fully arranged, i.e., both the treble and bass clef present

2. Have chord symbols

3. Composed by a recognised jazz musician

4. Arranged by a reputable user with a peer reviewed score of 4.7 stars out of 5 or above

\*\* Maybe insert table showing composers?\*\*

The solos are in MusicXML format, which is an XML-based format. A detailed list on the dataset can be found at Appendix.X.

3.3 MusicXML

In order to understand section 4, a brief explanation of MusicXML is provided in this section. For a more detailed understanding, see Good (2001).

MusicXML is an XML based markup language for representing musical scores. Like MIDI, MusicXML is standardised, and can be interpreted by all major music notation software and transformed into a score.

Figure 4 below shows how the first chord symbol and voicing pair in figure 3 is represented in MusicXML format.

Looking at figure 4, the chord symbol is represented by the use of a harmony element. The inner elements indicate the root of the chord and the kind of chord. The harmony element can contain additional elements for more complex chords.

The notes that make up the chord voicing are represented by the note elements that follow. The “default-x” attribute is helpful in determining which notes make up a chord voicing, as chord notes are always aligned vertically.

<harmony print-frame="no">

<root>

<root-step>C</root-step>

</root>

<kind text="M7">major-seventh</kind>

</harmony>

<note default-x="92.47" default-y="-40.00">

<pitch>

<step>E</step>

<octave>4</octave>

</pitch>

<duration>4</duration>

<voice>1</voice>

<type>whole</type>

<staff>1</staff>

</note>

<note default-x="92.47" default-y="-30.00">

...

</note>

<note default-x="92.47" default-y="-130.00">

...

</note>

<note default-x="92.47" default-y="-110.00">

...

</note>

<note default-x="92.47" default-y="-100.00">

...

</note>

Figure 4. A C Major 7th chord symbol and notated voicing represented in MusicXML format

4. Chord Scraper

This section will detail the chord scraper system developed and used within this project. A high level overview of the system will be provided as well as a detailed report of how the system was developed. The results and effectiveness of the system are presented in section 5.1. Sections 3.2 and 3.3 are recommended precursors to this section.

4.1 Requirements

- To extract pairs of chord symbols and the group of notes that make up the chord voicing from MusicXML solo piano scores.

- To output raw data as well as data that has been partially cleaned and refined (see section 4.1.2).

- To work on all solo piano scores in MusicXML format including previously unseen scores.

-To be able to scrape a large number of scores (>100) in a reasonable amount of time (< 5 minutes).

- To extract the group of notes that represent a chord symbol with a high degree of accuracy ( > 97%).

- To have a highly usable command line interface that enables other researchers to use the tool.

4.2 System Overview

As seen in figure 5, the chord scraper takes a number of piano scores in MusicXML format, and outputs chord data in 2 different formats. The first output format represents the chords exactly as they were inputted, whereas as the second transforms the chords into a format that it more suited to machine learning model inputs. More information of this can be found in section 5.1.

Diagram

Description automatically generated

Figure 5. A system diagram showing the chord scraper system

In order to highlight the workings of the system, a small scraping example is presented below.

Figure 6 shows a 2-bar extract from a jazz piano solo as a score view and as MusicXML. Figures 7 and 8 demonstrate the result of passing the MusicXML from figure 6 into the chord scraper. In figure 7, the chord voicings are represented as both a list of note names as well as their associated note numbers (reference/glossary/footnote?). As seen in figure 7, the data is a true representation of its input. However, in figure 8, chords 1 and 3 have been transposed (glossary) to have a root of C. This then means there is no need to include root information or note labels.

Schematic

Description automatically generated with medium confidence

<measure number="3" width="280.43">

<harmony print-frame="no">

<root><root-step>G</root-step></root>

<kind text="m7">minor-seventh</kind>

</harmony>

<note default-x="29.66" default-y="-15.00">...</note>

<note default-x="29.30" default-y="-145.00">...</note>

<note default-x="29.30" default-y="-135.00">...</note>

<note default-x="29.30" default-y="-125.00">...</note>

<note default-x="29.30" default-y="-115.00">...</note>

...

<harmony print-frame="no">

<root><root-step>C</root-step></root>

<kind text="7">dominant</kind>

<degree>

<degree-value>5</degree-value>

<degree-alter>-1</degree-alter>

<degree-type>alter</degree-type>

</degree>

</harmony>

<note default-x="153.20" default-y="-20.00">...</note>

<note default-x="153.20" default-y="-145.00">...</note>

<note default-x="153.20" default-y="-135.00">...</note>

<note default-x="165.07" default-y="-130.00">...</note>

<note default-x="153.20" default-y="-120.00">...</note>

...

<harmony print-frame="no">

<root><root-step>F</root-step></root>

<kind text="7">dominant</kind>

</harmony>

<note default-x="246.05" default-y="-25.00">...</note>

<note default-x="246.05" default-y="-150.00">...</note>

<note default-x="246.05" default-y="-140.00">...</note>

<note default-x="246.05" default-y="-130.00">...</note>

<note default-x="246.05" default-y="-120.00">...</note>

</measure>

<measure number="4" implicit="yes" width="180.55">

...

<harmony print-frame="no">

<root><root-step>C</root-step></root>

<kind text="7">dominant</kind>

<degree>

<degree-value>5</degree-value>

<degree-alter>1</degree-alter>

<degree-type>alter</degree-type>

</degree>

</harmony>

<note default-x="71.49" default-y="-55.00"></note>

<note default-x="71.49" default-y="-40.00"></note>

<note default-x="71.49" default-y="-30.00"></note>

<note default-x="71.49" default-y="-165.00"></note>

<note default-x="71.49" default-y="-130.00"></note>

...

Figure 6. A 2-bar excerpt of a jazz piano solo full arrangement as a score view and as MusicXML

root, type , extensions , note\_numbers , notes

1 G , minor-seventh, "[]" , "[23,26,30,33,52]", "[G2,Bb2,D3,F3,C5]"

2 C , dominant , "[{'degree':'9','alter':'-1','type':'add'}]", "[22,26,28,32,50]", "[Gb2,Bb2,C3,E3,Bb4]"

3 F , dominant , "[]" , "[21,25,28,31,49]", "[F2,A2,C3,Eb3,A4]"

4 C , dominant , "[]" , "[16,28,38,44,48]", "[C2,C3,Bb3,E4,G#4]"

Figure 7. Raw chord data representing score in figure 6 (CSV format)

type , extensions , note\_numbers

1 minor-seventh, "[]" , "[16,19,27,26,45]"

2 dominant , "[{'degree':'9','alter':'-1','type':'add'}]" , "[22,26,28,32,50]"

3 dominant , "[]" , "[16,20,23,26,44]"

4 dominant , "[{'degree':'5','alter':'1','type':'alter'}]", "[16,28,38,44,48]"

Figure 8. Cleaned and transposed chord data representing chords in figure 6 (CSV format)

4.3 Development

The chord scraper system was developed using the Python programming language. The reason for selecting Python was both its effectiveness as a scripting language and its array of inbuilt data manipulation packages.

The initial aim when developing the system was to write a script that could effectively and consistently extracting chord information from MusicXML files. Following this, an additional script would be developed in order to transform the extracted data into a format that would be more suitable to the machine learning model used for chord generation, as well as for other potential users.

For the purposes of readability, these two scripts will be presented as separate entities, however it must be noted that in reality they are to some extent, interconnected, and therefore exist inside the same Python file. The chord scraper can be found inside the *chord\_scraper* directory at the root of the project.

4.3.1 Chord Extraction

The first functioning version of the script worked by iterating through each line in each file and using Regex to extract the required data. The script worked by first finding a harmony element and extracting the chord symbol information. The way in which MusicXML is formatted means that the note element after each harmony element is the top note of that chord. However, because MusicXML lists all of the treble clef (glossary) notes before the bass clef notes, finding the remaining notes of a chord voicing using the order of elements was not possible. Figure 9 highlights this issue.

Diagram, schematic

Description automatically generated

Figure 9. A two-bar score showing the line order of MusicXML elements

After further analysing the MusicXML markup syntax, it was discovered that each note element had a “default-x” attribute, which denoted the notes *x*-axis coordinate. By using the “default-x” attribute of the note directly after each harmony element, the other notes that made up that chord could be found. This is because notes that make up a chord voicing are always vertically aligned. The issue with this approach is that there are cases in which some of the notes in a chord are slightly offset on the *x* axis. This would mean that some notes were not included in chord voicings when they should have been. The most common cause of this is when there are 2 notes in a chord that are next to each other on the piano, and vertically aligning them would cause the notes to overlap, and thus be unreadable. Figure 10 highlights this issue.

Diagram, schematic

Description automatically generated

Figure 10. A score highlighting the notes selected to make up a chord voicing using “x-location” value.

In order to solve this issue, a concept of *x-axis deviation* was introduced to the chord extraction script. This allowed for notes that were within *x* amount of the “x-location” to be identified as part of the chord voicing. For example, if the first note element after a harmony element had a “x-location” value of 100, and the *x-axis deviation* was set to 10, then any note with an “x-location” value of between 95 and 105 would be extracted as part of that chord voicing. The results and effectiveness of this approach are explored in detail in section 5.1.

Although this first approach functioned as expected, iterating through it line by line using Regex seemed like bad practice as XML is a structured markup language. After some further research, it was clear that a much more effective approach would be to parse each MusicXML file into a searchable data structure (citation?). Python’s build in XML API – *ElementTree* was identified as a suitable tool, as it was able to transform XML into an ordered Tree data structure. This would allow for search operations to be performed in constant time [O(1)] as oppose to linear time [O(n)] when iterating through *n* number of lines per file.

The second and final major version of the script used *ElementTree* to transform each MusicXML file into an ordered tree data structure. Rather than going through each file line by line, the second version traversed through the tree, bar by bar. Each bars children were then searched to extract each harmony element and their associated note elements to form pairs of symbols and voicings. The MusicXML tree structure can be seen in figure 11. The technique of finding the note immediately after a harmony element and using its “default-x” attribute to find all of the other notes was again used in this version. However, as each bars children were indexed, the implementation was much cleaner.

Diagram

Description automatically generated

Figure 11. MusicXML in as an Ordered Tree Data Structure

4.3.2 Chord Data Manipulation / cleaning / transformation

After developing a functioning chord extraction script, some additional data manipulation functions were created in order to improve the output of the chord scraper. For example, as chords by definition must have 3 or more notes, the gathered chord voicings that had less than 3 notes were removed. Also, the notes of each chord voicing were additionally represented using note numbers, a system in which the numbers 1-88 represent each note on the piano (reference). Outputting chord voicings in this way would allow other users to easily encode the data for further use. The chord voicings were also subjected to a small degree of pitch normalisation, in which any very high or low notes were moved further towards the middle.

Finally, all of the chords were transposed so that their root note was C. For example, every note in a D major-seventh chord would be moved down 2 notes (figure? Or more explanation). This then meant that there was no distinction between root notes, and only between chord types. For example, rather than having 200 C major-sevenths, 150 E major-sevenths, 400 F major-sevenths, and 250 Ab major sevenths, the data would consist of 1000 major-sevenths. Therefore, when using the dataset to train a machine learning model, there would be much more examples of each chord type. There is also no downside to this approach, as any generated chords can be transposed back to the root note of the user’s choice.

The decision to not perform some of these data manipulation steps when encoding the data for the machine learning model were as follows. The use of this chord scraper in the future by other researchers or users would most likely be for the purposes of extracting data for use in machine learning or data analysis. Therefore, it seemed like a sensible decision to output chord data in a state that would be desirable for such users. As noted in section 4.2, the chord scraper also outputs the data in raw format, before any data transformations have taken place.

Initially, the two datasets were outputted as data pickles (glossary). This was due to the fact that pickles make extracting and reimporting Python lists and dictionaries extremely straight forward. However many other programming languages do not offer in build support for data pickles (reference). Therefore, in order to make the chord scraper as widely accessible as possible, CSV format was used. It must be noted however, that storing Python data structures in this format is less than ideal.

The chord scraper also features a simple command line interface (CLI). Users can specify an input and output path, as well as enable error logging and the printing of chord meta information. Further details of this can be found here: (github link).

For an evaluation of the results and effectiveness of the chord scraper system, see section 5.1.

5. Results

5.1 Chord Scraper

Results

This paper was published in 2008 and presents a traditional programmatic solution to the task of lead sheet arrangement. The system takes as input chord symbols, and outputs a series of notes (voicing) for each of the chord symbols. The system is preprogramed with a set of jazz harmony rules, and voices each chord based on a series of conditional statements.

The resulting chord voicings were compared with *Band in a box (BIAB)* voicings in a percentual comparison test, in which the findings showed that the proposed system yielded a more preferable output.

Although this system does not employ the use of deep learning, its results will be a useful benchmark for which to compare the results of this dissertation research project.

Some of the latter parts of the system also provide an aid on how generated chords may need to be adjusted to fit with the melody of the lead sheet.

For example, the presented system ensures that the highest note in the generated chord does not surpass the melody note that is played alongside it.

Table

Description automatically generated with medium confidence

Classification model notes

Represented the data in 3 different ways and used RNN and CNN to run classification tasks

Report results…

Look at where label classifications were different that expected -> are they in any way more accurate? Could this potentially be used to classify chords without chord symbols in XML files?? To gather more data??

GAN notes

**Embedding:** transform indexes into a vector of fixed size. E.g., representing words as numbers

**One-hot embedding:** create a “sparse” binary vector that is the length of the dictionary (100 words = 100 length vector). Then each word is represented by a 1 instead of a 0 in the vector.

\_\_\_\_\_\_\_\_\_\_\_

https://www.youtube.com/watch?v=BUNl0To1IVw&t=3018s:

Generative models:

Autoencoders and Variational Autoencoders (VAEs):

Learn lower-dimentional **latent space** and **sample** to generate input reconstructions

GANs:

Don’t explicitly model density, and instead just sample to generate new instances

When generator is trained – it is learning to transform from distribution of noise to target data disttibution

Gaussian noise 🡪 target data manifold

Z 🡪 Y

Take on point from latent noise distribution – will result in particular output in target data space.

The points in latent noise distribution represent different outputs

If we transverse in the Gaussian noise space we get interpellation of output (2 outputs mixed together)

55:10 – conditional GAN

Introduce pairs to G and D

Paired translation

12 vector integer notation -> 88 vector full representation

Effective distribution transformers

\_\_\_\_\_\_\_\_\_\_\_

**How c\_gan works exactly:**

The generator takes in a label and outputs a prediction

Label -> **Generator**  - > Fake Chord

Label + Fake Chord || Real Chord -> **Discriminator**  - > Real or Fake

The discriminator takes in pairs of labels + fake / real occurences and outputs a value on whether or not they are real

After the model is trained we give the generator labels and it gives us predictions

Adapted pix2pix model:

Transformed chord notes and label data into 7x12 matrices (omitting bottom 3 notes and top note)

Adapted pix2pix model to contain 1D layers as oppose to 2D layers (see paper)\

Initial results:

The loss of the discriminator on both real and generated data rapidly went to near zero, why? Probably the generator

**Novel model**

“Both G and D could be a non-linear mapping function, such as a multi-layer

perceptron.”

**Discriminator**

Discriminator that takes in concatenation of labels and either fake/real chords

How should this data be represented/concatenated?

Look at results of classification tasks

When concatenating label and chord, should an extra dimension be added?

Discriminator should have a single output with sigmoid activation (value between 0 and 1 indicating whether or not it is fake/real)

If using 1d representation, should concatenation be on a third axis, i.e. (88,) + (88,) 🡪 (1, 176) **OR** on the x axis, ie. (88,) + (88,) 🡪 (176,)

How can this decision be made? Other than just testing it, is there some reasons for why one would perform better than the other?

Probably the third axis, as it represents the relationship between label and chord in a more meaningful way – EXPAND ON THIS

If using 2d representation of labels/chords -> what is the use of convolutions? Can they actually find patterns -> research this

**Generator**

Takes in a label and transforms it into a fake chord candidate

Again, do we need to use convolutional layers? Or will deep/fully connected layers do the job

Real examples = (label + real chord) + [1] (label for discriminator)

Fake examples = (label + fake chord) + [0] (label for discriminator)

**Training**

**GAN Loss Function:**

the generator tries to minimize the following function while the discriminator tries to maximize it:

* D(x) is the discriminator's estimate of the probability that real data instance x is real.
* Ex is the expected value over all real data instances.
* G(z) is the generator's output when given noise z.
* D(G(z)) is the discriminator's estimate of the probability that a fake instance is real.
* Ez is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
* The formula derives from the [cross-entropy](https://developers.google.com/machine-learning/glossary#cross-entropy) between the real and generated distributions.

(<https://developers.google.com/machine-learning/gan/loss>)

**Conditional GAN Loss Function**

**Backpropagation**

**Generator**

**Results:**

**Data Embedding**

I experimented with the data embedding:

**1)** 2 different 1D representations:

**1.** Source chord, real chord = (12,)(88,)

**2.** Source chord, real chord = (88,)(88,) [source chord(label) embedded to full 88 note vector in the middle C octave

**2)** a 2D representation:

(88,)(88,) 1D embeddings reduced to (84,)(84,) by removing bottom 3 notes and top note. Then transformed to (7,12)(7,12)

**Classification Models**

In order to determine the best embedding for the GAN, as well as the best type of network for the discriminator and generator, performed some classification tasks using all the embeddings and a combination of RNN and CNN.

1D RNN classifier using both **1.1 and 1.2** (above)

2D CNN classifier using **2**

Results:

Both performed incredibly well:

Give functions, data, graphs, etc

**Model 1: 2d pix2pix model (adapted for my dataset):**

Results showed that the discriminators loss quickly went to near zero – meaning that the generator was not generating convincing chords from source chords.

Show graph of loss over time and any other graphs/chards that were in 1D paper or other papers

Show some example generated chords

Why?

Unclear, using convolutional layers on simple binary chord matrices potentially didn’t yield any meaningful patterns

patchGAN uses a segment of the real/fake images (chords) passed in. ie. If matrices is 128x128, it looks at a 16x16 segment or multiple segments

this approach wont be as effective for chord data, as there would be notes spread around the vector, particularly on the x-axis (0 axis??)

Also mention loss functions /learning rate etc

**Model 2: 1D pix2pix model (adapted for my 1D dataset):**

Results?

Same as above

**Model 3: 1D Novel Model**

This model was born out of some initial testing using novel classifier networks

**V1**

The discriminator had a single output neuron with sigmoid activation with binary-crossentropy loss function

Results…. Loss of the discriminator on fake chords quickly reduced to 0, meaning generator was not improving

\*mention the loss function of the GAN here\* - explain it etc

\*more results here with graphs etc and chords shown in images etc with stats\*

**V2**

Created an 88 neuron output layer on the discriminator

\* mention how this changes the way the loss function works in the c\_gan model

Discriminator loss on real chords went from 1.5 – 1.2 over 11 thousand iterations of training

Loss on fake chords quickly went to zero

**Changed way source chord and real/fake chord concatenated**

Initially it was (88,)(88,) 🡪 (2, 88)

Maybe dense layers don’t perform as well on matrices?? Reference/research

Changed to (88,)(88,)

**Leadsheet XML Arranger**

**INPUT:** leadsheet (musicxml)

Step 1. Extract chord symbols

Step 2. Embed chord symbols as 88 “source-chord” note vectors (see preprocessing chapter)

Step 3. Pass source chord vectors into c\_gan model

Step 4. Combine leadsheet melody with generated chords into full arrangement

**OUTPUT:** Chords + Melody Arrangement

Data parsed from lead sheets

**Meta information:**

-Key

-Time signature

**Chords:**

-Chord symbol

-Bar number

-Position in bar

-Note at chord position (won’t account for 2 voicings / not sure which note it will get if chord present)

Data Issues

The number of different types of chords in the data was skewed :

Insert pandas table

1000 dominant 7 wheras much less of other chords

Data skewness

Data should be bell curve

But currently it is right skewed

Something to do with mode median

SOLUTION?

Get more data + remove some label occurences

WHEN SPLITTING TEST vs TRAIN:

INITIAL DATA COUNTS WITHOUT WIDENING CHORD SELECTIONS:

dominant 1177

minor-seventh 1053

major-seventh 674

major 453

dominant-ninth 255

suspended-fourth 249

dominant-13th 159

minor-ninth 133

half-diminished 129

minor 117

major-sixth 96

major-ninth 68

minor-11th 63

diminished-seventh 59

diminished 44

minor-sixth 38

Name: label, dtype: int64

Performing data analysis using pandas and matlabplot

Removed all chord labels that have less than 600 occurrences

Created a stacked bar chart representation of each label to analyse data and note occurrences

\*\*insert bar charts\*\*

For the large part the data was good - \*\*insert\*\* margin of error numbers

However a small number of labels had wrong notes associated with them

\*\*insert\*\* some examples with numbers of wrong notes

Therefore further data cleaning was performed. Created array of unwanted notes for each label, and iteratively removed those notes from each chord vector.

\*\* insert cleaned bar charts\*\*

We now have

References

Ji, S., Luo, J. and Yang, X., 2020. A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions. arXiv preprint arXiv:2011.06801.

Hadjeres, G., Pachet, F. and Nielsen, F., 2017, July. Deepbach: a steerable model for bach chorales generation. In International Conference on Machine Learning (pp. 1362-1371). PMLR.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. Advances in neural information processing systems, 27.

Szelogowski, D., 2021. Generative Deep Learning for Virtuosic Classical Music: Generative Adversarial Networks as Renowned Composers. arXiv preprint arXiv:2101.00169.

Zhao, Y., Qiu, L., Ai, W., Shi, F. and Zhu, S.C., 2020. Vertical-Horizontal Structured Attention for Generating Music with Chords. arXiv preprint arXiv:2011.09078.

Szelogowski, D., 2021. Generative Deep Learning for Virtuosic Classical Music: Generative Adversarial Networks as Renowned Composers. arXiv preprint arXiv:2101.00169.

Dong, H.W., Hsiao, W.Y., Yang, L.C. and Yang, Y.H., 2018, April. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In Thirty-Second AAAI Conference on Artificial Intelligence.

Angioloni, L., Borghuis, T., Brusci, L. and Frasconi, P., 2020, November. Conlon: A pseudo-song generator based on a new pianoroll, wasserstein autoencoders, and optimal interpolations. In Proceedings of the 21th International Society for Music Information Retrieval Conference ISMIR MTL2020 (pp. 876-883).

Valenti, A., Carta, A. and Bacciu, D., 2020. Learning a latent space of style-aware symbolic music representations by adversarial autoencoders. arXiv preprint arXiv:2001.05494.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D. and Simonyan, K., 2017, July. Neural audio synthesis of musical notes with wavenet autoencoders. In International Conference on Machine Learning (pp. 1068-1077). PMLR.

Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

Good, M., 2001. MusicXML for notation and analysis. The virtual score: representation, retrieval, restoration, 12(113-124), p.160.