Musical Harmonisation via Machine Learning

CM2303 - One Semester Individual Project - 40 Credits

Cardiff University School of Computer Science and Informatics

Author: Brendan Rhys Lamb (C1610560)

Supervisor: David Marshall

Moderator: Matthias Treder

Abstract

The renowned American composer Leonard Bernstein once put forward the idea that music could be conceptualized linguistically through the analogy of syntax. This project seeks to investigate the validity of this claim by applying deep learning techniques and technologies commonly found within the field of natural language processing, such as language modelers and recurrent neural networks to the task of musical harmonisation. This project investigates the effectiveness of these techniques via comparison and evaluates their validity in their application in the field of music. The results of this study suggest that while certain techniques have limited practicality in this field, in particular the use of word2vec word embeddings and the use of long short-term memory neural networks show great potential for use in deep learning music processing.

Acknowledgements

I wish to give my sincerest thanks to David Marshall, my supervisor for this project, for remaining encouraging throughout the entirety of the project, even amidst a pandemic, and for keeping both my interest inspired and self-belief alive throughout the semester.

I would also like to thank my loving partner Alex, who has observed and assisted through many a late night of data crunching and report writing. Their support has been invaluable and irreplaceable.

Finally, I would like to thank my dearest friend Freya, with whom I have bonded over these three years. Many a story about the thrills and stresses of our respective projects has kept up my resolve.

Table of Contents

Table of Figures

1 - Introduction

Machine-written music has been a concept that might have seemed to be science-fiction a number of years ago, but today we have many examples of such music, many of which have been generated using machine learning systems. Perhaps part of the reason why music is such an interesting problem for machine learning to tackle is because of the inherent mathematical nature of harmony. For those reading who have no musical literacy, a short description of the musical knowledge required for understanding the concepts in this project will be provided in section 2.1, within the background section of this report.

Many of these machine-written pieces lack understanding or consideration for long-term structure of music, resulting in harmony that doesn't sound as natural as human-written music. Leonard Bernstein states in his 1973 series of lectures The Unanswered Question that music can be conceptualized through the lens of linguistics. In his own words: "It is in the nature of music to be ongoing".[10][11]

A field greatly benefiting from developments in deep learning is that of natural language processing, which exists at the intersection of computer science and linguistics. Natural language processing is an area of research that seeks to allow machines to understand and process human language.[19] Deep learning has allowed for the creation of new approaches to machine learning, such as word2vec, which was developed by Google in 2013. These approaches have found a home in the field of natural language processing and have allowed for the field to experience rapid development in recent years. Prominent technologies such as text classifiers and language modelers have made extensive application of such approaches.

The understanding of this project is that if Bernstein's thesis is true, this means that language and syntax is an effective proxy to allow us to understand how humans conceptualize music. In this case then using deep learning approaches commonly utilized in the field of natural language processing must also have practical application within the context of machine-written music.

The aim of this project was to investigate ways that music can be understood and processed by machines via the use of deep learning by utilizing aforementioned techniques and approaches found in natural language processing. This project will be

using harmonisation as a measure to assess how well each technique has successfully processed an understanding of music.

There are two primary reasons for harmonisation being selected as a metric of success for each technique. Firsty, correctness of harmonisation can explicitly be measured. Like a fluent English speaker can identify an incorrect placement of a verb in a sentence like “we together danced” or “I lunch bought”, a trained musician can point out an incorrect or *non-functional* chord within harmony.

The second reason is because well written harmonisations consider both melody and the context of the harmony itself. To use another language analogy, this is similar to an English speaker using context to complete sentences. A fluent speaker, when asked to complete a sentence fragment like “Yesterday, I ate ____”, would likely fill in that blank with an item of food. For a human musician, this ability requires a good understanding of music, and so the same can be assumed of a non-human musician. This project seeks to understand how effective the use of melodic context is when compared to harmonic context, and in addition how effective the use of either of these are when compared to a combination of the two.

In order to test natural language processing techniques, this project constructed three groups of machine learning models. Each of these groups utilizes a different technique or approach to achieve the goal of completing a harmonisation.

The first group of approaches seek to measure the accuracy of solely using melodic context by making use of a recurrent neural network to implement a character-based language model to predict which chords will be present in a bar of harmony when given a bar of melody. This model predicts chords based purely on melodic data, without the context of surrounding chords. The second group of models seek to measure the accuracy of using solely harmonic context by utilizing a long short-term memory neural network to train a continuous bag-of-words model that will predict chords purely given the surrounding chord context. The final group of models have the goal of utilizing a combination of harmonic and melodic context. This group of models used the same long short-term memory neural network as the previous group of models, however, this group utilized a different form of data preparation that allowed melodic data to be input into the model. Each group of models features a number of experimental differences that have been compared to determine what parameters or data preparations best suit a model or technique.

The intended application of the knowledge gained from this project is to assist musicians within the context of composing music and completing their own harmonisations. A large part of musical training focuses on developing the ability for a musician to understand the function behind harmony. Because of this, practical application for the knowledge discovered from this investigation can be found in a tool that gives a user a recommended chord in a given point of their progression, or perhaps even substitutions that can be made for that chord in order to help the musician make their progression more interesting. Such a tool might provide the ability for less trained musicians to learn about more complex harmonies. In addition, this knowledge would contribute to the existing knowledge base of the applications of machine learning in music. Future developers might have an easier time with additional data backing up the accuracy of certain techniques.

The dataset used for this project is a collection of Johann Sebastian Bach's music, which can be found as part of a group of available corpuses natively within the library *music21* which was utilized heavily within this project. This dataset was used as the simplicity of Bach's music is often a factor as to why Bach is taught to students. It was this project's hypothesis that this simplicity would allow for any deep learning algorithm to have less difficulty in the task of locating patterns within the dataset. This project does make the assumption that the dataset chosen is large enough to allow such a group of deep learning models to thoroughly process and understand musical data. This work could be vastly improved by the optimization of the dataset.

This project's scope is restricted to the creation of and gathering of data from these machine learning models. A discussion about knowledge learned from the creation of the models will take place during this report, as well as a discussion about what understanding can be gained about the implication of the application of natural language processing techniques within the field of music. However, the development of an application or graphical user interface to make use of these models lies outside the scope of this project. This project seeks to lay some level of foundation for such work and it should be considered either further research or an application of the knowledge gained.

The conclusion of this work is that the best approach out of the ones studied is that of an LSTM-based model that combines the use of melodic and harmonic context when predicting chords, as it leads to a model accuracy that far exceeds those found in the other two model groups, likely due to an LSTMs increased ability to retain information over a longer sequence of data compared to ordinary RNN.

2 - Background

2.1 - Musical Theory and Harmony

Understanding of this project and its goals will likely require at least a basic understanding of western tonal harmony. For the non-musically inclined, some basic concepts of harmony will be explained here, along with a brief description of their significance to the project.

In music, harmony is defined as the combination of simultaneous tones. We can combine notes from a melody with a number of other tones to create chords. This harmonises the melody. Below is an example of more than one instrument being used to create harmony in the key of F minor.

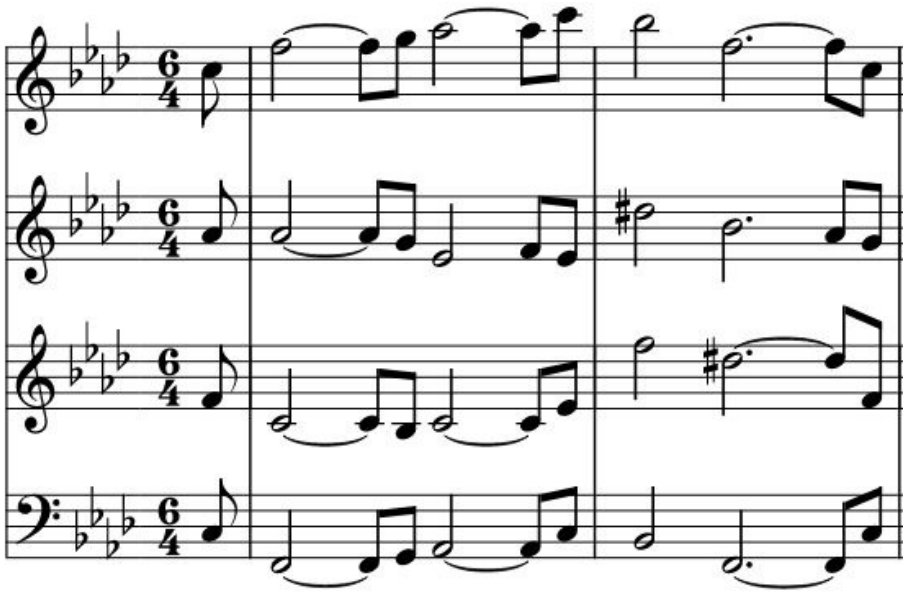


Figure 1: A short musical excerpt demonstrating the use of four-part harmony.

In some pieces, such as the one above, harmony is constructed by a number of individual instruments, or voices playing one note each at a time. In other examples, each of these notes might be played by the same instrument, as chords. Below is the same excerpt of music, but represented as chords, instead.

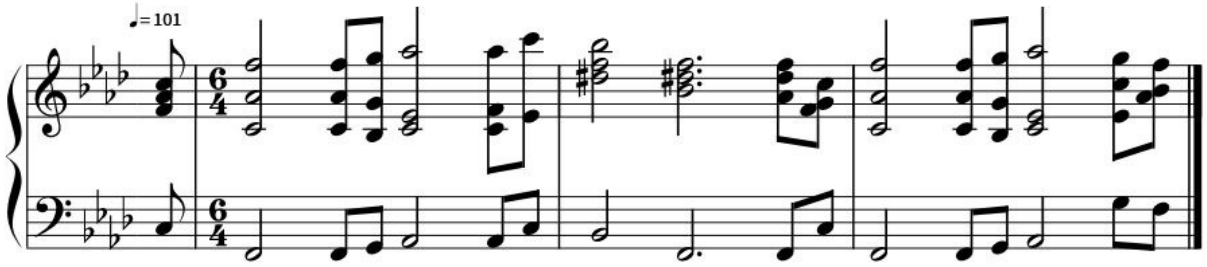


Figure 2: A short musical excerpt demonstrating the four-part harmony shown in figure one, condensed into chords

Typically, the foremost way to create harmony is by taking a look at what key we are in. A key is simply a group of pitches that a piece can use, somewhat analogous to a color palette. The key of D major for instance, uses the notes D, E, F#, G, A, B, and C#.

Each key typically has seven associated triad chords, one for each note in the key. Triads are an integral part of harmony, and in simple terms are a chord of three specific notes played simultaneously. The first note in constructing a triad is named the root. The root is the lowest note in the chord, and the note that lends the chord its name. The second note is named the third of the triad, and lends the triad its tonality, either major or minor. This is dependent on the number of notes, or semitones between the root and the third. If the third is four semitones higher than the root, it is a major third, and if it is three semitones higher than the root, it is a minor third. The last note in a triad is named the perfect fifth and it lies seven semitones above the root. For example, a D Major triad, would contain the notes D, F# and A. F#, the major third lies four semitones above D, and A, the perfect fifth lies seven semitones above D.

A simple way to construct triads is to look at what notes are available in a key. We will use the prior example of D major: D, E , F#, G, A, B, and C#. To construct a triad in any chord, we simply take our root, skip the next note in order, take our third, skip the next note in order, and then take our fifth. For example, we could take E, G, and B to create an E minor triad.

The seven triads of a key are often represented by roman numerals, I through vii. If the numeral is capitalized, the chord is major, else it is minor. For instance, A major is represented as chord V of the key of D major, as A is the fifth note in the D major key, and it is a major chord. E minor is represented as chord ii of the key, as E is the second note in the D major key, and it is a minor chord.

Given this brief description of triad chords and keys, it is now possible to harmonise a melody. To harmonise a melody, we need to select a section of melody to harmonise. In this example, we'll stay the key of D major. Our chosen section of melody contains the notes G, B, C# and D. To select a chord, the chosen chord should ideally contain notes from the melody within. In this example, we know that G major is a chord found in our key, D major, and the three out of four notes in our section of melody are also found in the chord G major. As such, this would be a good chord to use. A musician can then do this for other sections of their melody until the entire piece of music has been harmonised.

With that brief explanation of harmonisation, it should be noted that this is one small part of what a musician might choose to consider when harmonising their melody. Another aspect to consider is that of chord progressions. Each chord possesses some level of 'tension', as described by musicians. A chord with high tension is able to be resolved by following the chord with a related chord that releases said tension. Chord V in any key possesses a lot of tension, which is often released by following it with Chord I, which releases tension and creates resolution.

This understanding of tension and release has led to certain cycles, or progressions of chords being present throughout a lot of music, as tried and tested ways of manipulating tension. Some examples of common chord progressions are I - VI - V - I in classical music, the I - vi - IV - V progression in pop music, or the ii - V - I turnaround, in jazz. These are important to note, as an ability to predict some of these chord progressions provide an excellent insight into the effectiveness of the machine-learning model produced by this project.

This knowledge is of importance to this project, as this provides insight into how data will be prepared for model training.

2.2 - Prior Research into Deep Learning Music

Using a basic understanding of Western harmony as provided in section 2.1 to provide accompaniment to a melody, we could create something that sounds correct, but there is more to music than harmony and melody simply sounding correct. As mentioned, prior chords often play a part in chord choice, not dissimilar to a written story. Functional sentences, whilst correct, do not mean anything on their own. Leonard Bernstein noted in his 1973 series of lectures The Unanswered Question that similar ideas could be applied to music. Much like in poetry or prose, can give a chord new meaning depending on its context. Bernstein states in the same series of lectures, “It is in the nature of music to be ongoing”. As stated in my Initial Plan, these lectures by Bernstein serve as my primary motivator for this project. Bernstein puts forward the idea that music could be conceptualized as a language. Bernstein notes that ideas such as semantics and syntax can be applied to music.[10][11]

Bernstein himself was a renowned conductor and composer, and his ideas were controversial in the musical sphere at the time. Given age, however, these ideas have been the subject of research for musicians, neuroscientists and psycholinguists. Since Bernstein delivered these lectures, a number of papers in these fields have been published, delving into music with this approach.[3]

It is because of these ideas that this project seeks to understand how machine learning can be used in harmonisation in this manner. Techniques for language processing are abundant within the realm of machine learning already.[4] Taking into account the ideas of Leonard Bernstein, this project sought to understand the effectiveness of these pre-existing language modelling techniques towards solving the problem at hand.

Magenta is a project by Google that has looked into similar approaches before. According to Google, Magenta is an open source project that seeks to research more about using machine learning as a creative tool.[5] Through the use of Magenta, Google has published a number of papers in regards to the effectiveness of certain approaches in regards to music generation via machine-learning.[1][5] Analysis of these approaches in regards to music generation allowed this project to determine which approaches might be applicable to harmonisation.

In the publication Approachable Music Composition with Machine Learning at Scale, researchers designed an application that inputs a melody, and outputs a harmonisation

for that melody. This was achieved using Coconet, a predictive machine learning model which was trained on Bach Chorales. Coconet was trained through the process of randomly erasing notes from these chorales, and then asking the model to fill back in the chorales using the context of prior and previous notes.[8][9]

While the work done by Google focuses on harmonisation of a melody, along with generation of music, this paper demonstrates that a similar approach was applicable to the training of the models developed in this project. Coconet uses one-hot encoding to encode pitches, and then uses those encodings to build a predictive model.[5] Approach A and Approach B in this project however use one-hot encoding on chord objects. Arrays of encoded chord objects then used to represent the entire piece of music.

Long short-term memory neural networks are a very common feature of these projects, and for good reason. Ordinary recurrent neural networks struggle due to their inability to retain information over long sequences of information. This can cause issues in applications such as natural language processing or music processing, due to the inherent sequential nature of the data used. The answer to this problem is the LSTM neural network.

This long-term memory can be highlighted in the practical examples of LSTM neural networks. In their 2002 publication Learning the Long-Term Structure of the Blues, Douglas Eck and Jurgen Schmidhuber noted the lack of musical structure in music composed by the use of RNNs.[1] In an attempt to create a neural network that could create machine-written music that possessed such structure, their experimentation noted that the use of LSTM layers in their network instead of basic RNNs allowed for the production of melodies and structures that possessed some level of musical competency.

2.3 - Natural Language Processing

Natural language processing is an area of research that seeks to allow machines to understand and process human language.[19] A considerable amount of data preparation goes into deep learning algorithms used for natural language processing. Some of these data preparation techniques will be used within this project.

Tokenization is the process of breaking down an input text, or corpus into individual words. Certain characters are removed as necessary in an effort to make the text machine-readable.[19] A translation to music can be assumed if we consider a piece of music as “text”. It would be possible to tokenize this music into its constituent parts such as chord or notes in order to make this music machine readable.

Word embeddings are a way in which words are encoded to make them machine readable. A common approach to encoding words in this manner is word2vec. Word2vec was patented by Google in 2013 and allows for words to be encoded as n-dimensional vectors. This allows for the words to be mapped within dimensional space. Similarities between words may be interpreted as their close proximity within one or more of these dimensions [6][14][23]. This too may be translated to music. If we consider that any chord might imply another chord is to follow it, these chords can conceivably be mapped using word2vec.

Two fundamentally distinctive word2vec architectures exist. That of the continuous bag-of-words model, and that of the continuous skip-gram model. Both of these approaches are utilized in the development of language modelers, but possess differing functions. A continuous bag-of-words model will attempt to predict a target word when given surrounding context words, whereas a continuous skip-gram model will suggest context words when given a target word. [14]

Long short-term memory neural networks are also utilized heavily within the field of NLP. LSTM networks are an innovation built on the foundations of Recurrent Neural Networks, and are explained in more detail in section 2.2 of this report.

2.4 - Dataset Utilized

This project, as with all machine learning based projects, needed to make use of a suitable dataset that could be used to both train and evaluate trained models. Most musical data is available in audio formats such as .mp3 or .wav, but while these formats are perfectly acceptable to a listener, they do not contain any sort of musical information about the musical pieces they contain. One could not open up a mp3 file of their favourite song, and learn what notes are being played, for instance.

This was an issue that had to be overcome. As stated, this project sought to conceptualize music and harmony as a language, and so training a model on audio data would be equivalent to training a language classifier based on audio. Potentially useful, but something that would result in a complexity that is out of the scope of this project.

In order to make the goals of this project achievable, some format that allows for raw musical information to be accessed was necessary. One such file format was immediately obvious, MIDI. The .mid file format is used to save MIDI sequences, usually for the purpose of saving composed music tracks, or for writing scores. MIDI saves note values in sequence. One can open up a .mid file, and at any point in the duration of the file learn which notes are currently playing. This allows for music to be analysed in a much more machine-friendly manner.

In addition, a second file format was also under consideration, MusicXML or .mxl. MusicXML is a form of XML that was specialized for the use of music scoring applications such as MuseScore and Sibelius. MusicXML also possesses capabilities for representing chords as their own object, as opposed to MIDI. This means that using MusicXML allows for the extraction of chord data without having to construct those chords from individual notes, as if we were using MIDI data.

Initially this project chose to make use of a .mid dataset consisting of many jazz pieces. However, there were a number of problems found with this dataset. Firstly, some of the files contained data for percussion tracks. Percussion tracks in midi utilize different note values to represent different percussive sounds. The note A4 might represent a snare drum for instance. This would obviously lead to erroneous output if used to train a model. In addition, a lot of the midi data used in the dataset was captured from live piano performance, which meant rather than notes being quantized being neatly they instead fell at the exact moment the pianist pressed each key.

These reasons are why the project eventually made use of an MusicXML. MusicXML files are available in large quantities as scores, which allow music to be quantized in a much better manner.

The musical dataset that ended up being selected for use within this project was a dataset found native to Music21, a musical toolkit library for Python which is discussed in section 2.5 of this report. This dataset is that of a corpus of Johann Sebastian Bach's chorales. The reason for this is that Bach's music is famously mathematical, and so this project hypothesizes that Bach's use of mathematical patterns in his work will allow a machine learning algorithm to have an easier time recognizing patterns within the music.[25].

2.5 - Music21

Extraction of musical data from MusicXML or MIDI isn't a feature that comes native to Python. In order to make use of the dataset, some form of toolkit is necessary. There isn't too much competition in this regard, and Music21 stands out clearly as a contender for the standard of musical feature extraction.

Music21 is a library for Python developed by MIT. It includes a number of features essential to this project. It is able to use several file types as input, including MusicXML, and represent them as a Stream object. A Stream is somewhat analogous to a score, and consists of elements such as notes, chords, key signatures, bars. These features are then able to be used in code.[27]

Each of these elements is able to be extracted. Note objects and chord objects saw the most frequent use within this project, understandably. Where possible however, these features were encoded into a more machine-readable format, and as such these elements were only ever directly used within the data preparation section of model development.

Music21 allows these features to be of use in the Python program. As mentioned earlier in the Harmony section of this chapter, there are a number of things we are interested in. Namely, we can use this tool to identify the melody of the piece, and the harmony of the piece. An exact description as to how this feature extraction is achieved is available in section 3.2 of this report.

2.6 - PyTorch

As opposed to the search for a musical extraction tool, the pool of machine learning libraries to select from offers much more choice. The largest two contenders were between PyTorch and Keras.

Keras is a machine learning API that is built on top of TensorFlow, which is another machine learning API. While TensorFlow is quite low-level in terms of abstraction, Keras offers a higher-level approach than its PyTorch alternative, and according to some is easier to use when trying to get short-term projects started. However, this simplicity can make working with Keras inflexible at times.[2]

PyTorch however offers a midrange approach in terms of abstraction between using TensorFlow and Keras, both being user-friendly and flexible to some degree at the same time.[2] Documentation for PyTorch appeared to be more plentiful too, which seemed to be something that would aid the project greatly if it ran into any problems. In addition to these points, PyTorch has better training speeds and performance than Keras. As this project would make use of my home computer and its graphics card, this seemed like a very justifiable reason to select Pytorch over Keras.

As such, the combined flexibility, relative ease of use compared to other lower-level APIs and its strong performance are all factors which led to this project's use of PyTorch.

3 - Solution Overview

3.1 - Core Objectives and Models Produced

This section will discuss how the task was broken down into smaller objectives to shape the task into something more achievable.

After initially considering the background research for this project, three core objectives were laid out during the initial report for this project. Originally, the purpose of these three core objectives were to provide some structure towards the development of a neural network, but over the course of the project this purpose had shifted slightly. The original three core objectives are listed below.

Core Objective 1: Basic Chord Prediction

"The first core objective is that of a basic neural network, and for that neural network to be capable of basic chord prediction. To achieve this milestone, the system should be able to take a melody as an input, and output harmonically functional chords to accompany the melody as an output. Via training, the neural network should be able to achieve a level of functionality that will allow it to suggest chords per bar."

Core Objective 2: Looking at Structure

"The second core objective will be achieved when the neural network is able to analyse the overarching structure of the music. In order to do this, the neural network will need to be modified to become an LSTM neural network. Via an approach such as word2vec, which has shown promise in analysing musical data, the neural network should possess the tools it needs to analyse the music. A success on this milestone will result in a neural network that uses information from other bars in the music to make choices about chords. Use of cadences in correct positions would be a sign that this has been achieved."

Core Objective 3: Improving Structure

> *"The third core objective will be achieved when the neural network makes use of motif in its choice of chords. The neural network will be required to search the given melody for patterns. These patterns represent a motif in the melody. If a motif occurs once at the start of a passage, and then again at the end of the passage with a slight alteration, the network's choice of harmony should reflect this."*

These three core objectives, while a good start for this project, were incredibly broad in terms of what this project would be trying to investigate and learn. Instead, these three core objectives were used as a starting point to lay out three distinctive methods for harmonising music via machine learning, each featuring a contrasting approach to the problem of harmonisation. These models are being used to compare the effectiveness of these methods after the conclusion of experimentation.

Before delving into how data was prepared for each method, and how each group of models was constructed, it would be greatly beneficial to elucidate the aims of each method, how these relate back to the aims of the project as a whole, and can be learned from a comparison of these models. While the overall goal of these three models are the same: to predict chords given some level of information, the way in which they utilize their given data is what differentiates them.

The first method was that of the RNN Method. This was based on the prior Core Objective 1 and utilized perhaps the most straightforward approach out of the three models. It sought to tackle the problem of harmonisation solely by making use of melodic data in its predictions. Because the RNN Method does not make use of any sort of context chords or harmonic context, this is an excellent control measure when comparing it to methods Two and Three, when trying to determine how effective the use of harmonic context is in relation to harmonisation, as both later models utilize such harmonic contexts. The goal of the RNN Method was to produce a model that when given a series of melodic notes,will predict the most harmonically appropriate chord to harmonise those notes. The solution for this method consisted of a recurrent neural network, or RNN, that predicts a complete bar of harmony for a given bar of melody.

The second method, which this project has labelled as the LSTM Method, was adapted from Core Objective 2's premise of utilizing word2vec and LSTM layers in a deep learning model. The LSTM Method's intention was to understand the structure of music

by basing harmonisation solely on the context of existing harmony. The idea behind this method was for the produced model to predict a target chord based on the chords surrounding it. Contrary to the RNN Method, this model utilizes no melodic context whatsoever, which allows for an adequate comparison in the effectiveness between the use of harmonic context and melodic context when harmonising via the use of machine learning. In The LSTM Method, chords are represented as a roman numerals. This allows for simplification of the data being processed for input into the model, allowing for faster training times. It also allows the model to utilize data from any musical key in a useful way.

The third method, which has been designated as the Improved LSTM Method, in part utilized the sentiment from Core Objective 3, that for a model to demonstrate strong musical ability, it should prove that it understands some conceptualization of motif. Whilst this Core Objective may have sounded feasible to the novice to Machine Learning that I was at the time of writing, determining what does or does not count as motivic data was incredibly difficult to define. Failing to find an explicit answer for this question, it was decided that a combination of rhythmic and harmonic data seemed to be a good approximation of motif.

As such, the Improved LSTM Method sought to combine the previous two methods by making use of both melodic and harmonic contexts. The Improved LSTM Method had the aim of predicting target chords, given a combination of melodic context and harmonic context. The way this was achieved was through the use of a specific function that will be discussed in a later section of the report, that allowed melodic and harmonic context to be combined into the same data object. This allows for some level of melodic shape to be preserved in the training for the neural network. Unlike the LSTM Method, chords are not represented as roman numerals, but are instead represented as a string consisting of the chord's pitch classes.

To simplify, in order to conduct this investigation, three groups of models have been constructed. Each group utilizes a unique common technique found in the field of natural language processing, and makes use of different types of musical data, to determine their importance when considering harmonisation via machine learning. The RNN Method makes use of melodic context in order to create harmonisations, the LSTM Method makes use of harmonic data in order to predict harmonisations, and the Improved LSTM Method makes use of a combination of both melodic and harmonic data to predict harmonisations.

3.2 - Data Preparation and Feature Extraction

By an exceedingly large margin, preparation of the dataset was the part of this project that required the most work. Extraction of features from musical data found within the datasets formed the backbone of this project. Discussion about the choice of dataset, as well as problems and solutions found with said datasets, are discussed in the background section of this report, at subsection 2.3.

A brief discussion of the function that was originally used to extract features from the musical data that was used for the majority of the project is necessary. After a period of use, it was eventually found out to be strongly limited, and an alternative feature extraction function had to be devised. This original function should be criticised to highlight the benefits of the new improved feature extraction function.

```
def streamInput(input = None):

    if (input == None):
        works = corpus.getComposer('bach')
    else:
        works = input

    harmonyWorks = []
    melodyWorks = []
    keys = []
    for work in works:

        harmonyBars = []
        melodyBars = []
        k = []
        score = converter.parse(work)

        if (len(score.parts) > 2):
            length = (len(score.parts[0].getElementsByClass('Measure')))

            #Grab individual bars
            cc=score.chordify(addTies=False)
            measures = cc.getElementsByClass('Measure')
            for i in range(0, length):
                measure = measures[i]
                harmony = []

                cs = measure.getElementsByClass(chord.Chord)

                for c in cs:
                    harmony.append(c)
                harmonyBars.append(harmony)
                k.append(measure.keySignature)

                melody = []
                for n in score.measure(i).parts[0].pitches:
                    melody.append(n)
                melodyBars.append(melody)

            harmonyWorks.append(harmonyBars)
            melodyWorks.append(melodyBars)
            keys.append(k)
    return [harmonyWorks, melodyWorks, keys]
```

Figure 3: A screenshot depicting the disused segment of code for the function *streamInput, once* utilized for feature extraction early into the project's lifespan

Above is the original function that was originally used to extract features from musical data, lovingly labelled *streamInput*. This feature extraction function prompted for a list of .mxl files as its input argument. If no files were given, the function defaulted to Johann Sebasian Bach's chorales as a dataset. This data was then split into individual pieces of music or works. Each piece of music was then broken down into measures.

Before anything was done to the music, it was required to separate the music into two distinctive groups: melody, and harmony. Irritatingly, as with a lot of more basic musical concepts, there exists no explicit definition as to what a melody is, so an assumption was made that the melody being played in a piece of music was the highest-pitched voice or instrument. The music of this highest-pitched instrument was isolated and defined as the melody.

Within the Music21 toolkit exists a function that allows a function to condense all notes occurring during the same moment in time into a single chord. This was used in order to grab all non-melody notes and condense them into chords, if they were not already. This allows every piece of music to have an explicit harmony section.

The music at this point was evenly divided into a melody section and a harmony section. What was required now was to loop through each bar and extract the required elements from each section. In each measure of both the melody and harmony, a number of features were extracted from the music. Every chord in the measure was stored in a list labelled *harmony*, and every melody note in that measure was stored in a separate list labelled *melody*.

Then, these lists were stored in two separate lists respectively containing every measure in each work. These were labelled *harmonyBars* and *melodyBars*. A key signature for each work was saved in a third list labelled *keys*. Finally, the list of all measures of harmony and melody were added to further respective lists, representing each work. This was so that we can use the syntax *melodyWorks[2][3]* to get the third bar from the second available work in the list.

This method for processing data input resulted in the data still requiring more preparation later down the line, but left it in a format that allowed it to be easier to work with. Rhythm doesn't play much part in harmonisation, so at the time this was not seen as a necessary feature to extract, an assumption that caused a lot of problems when trying to design a suitable feature extraction function. Using this feature extraction function, outputs were three lists, one of which contained every chord in the sequence

in which they appear within the music, the second containing every melodic note in the sequence in which they appear in the music, and finally the third which contained a key for each piece of music.

So what caused *streamInput* to stop being useful? While this feature extraction function would have been useful for certain potential applications, it was missing a crucial ability to tell how long each chord or note was being played. To show why this would have been a large complication, the problem tackled in the RNN Method will be considered. In the RNN Method, the goal was to create an RNN that could predict chords for a given section of melody. We wish to train the model to predict harmony using this musical input as an example:

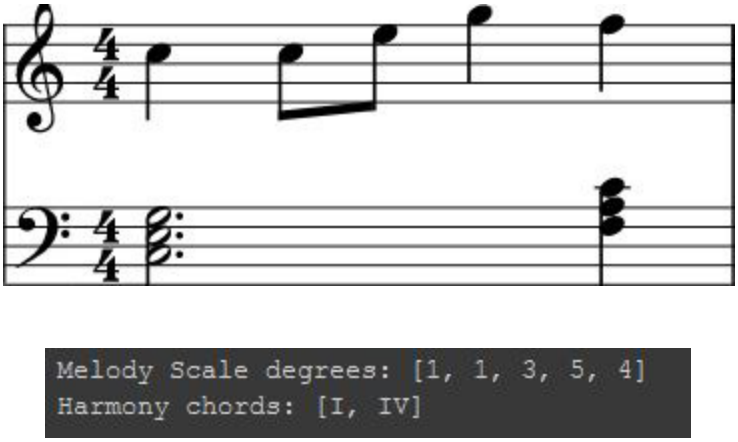


Figure 4: A short except of music along with its representation in scale degrees and chords after being processed by the original *streamInput* function

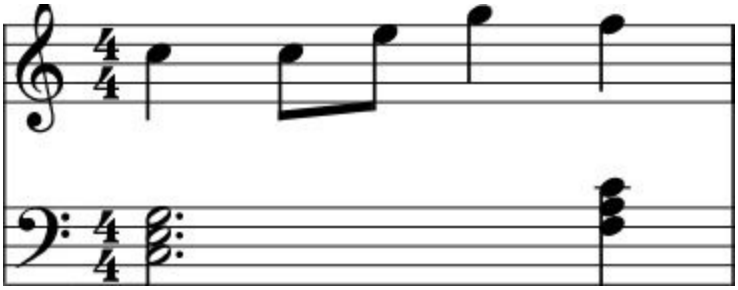
After processing, the output would offer little information to distinguish which sections of the melody were playing over which sections of the harmony. In this example of prediction, this would have been extremely problematic. How would we have any idea which chords were supposed to be playing to accompany scale degree 3 for example. As such, a different feature extraction function was required.

```
def streamInput(input = None):

    #Check if we have any params
    if (input == None):
        works = corpus.getComposer('bach')
    else:
        works = corpus.getComposer(input)

    mo = [] #Melody Output
    ho = [] #Harmony Output
    ko = [] #Key Output

    #For each piece in dataset
    for work in works:

        #Read the piece and set some variables
        score = converter.parse(work)
        if (len(score.parts) > 2):
            mi = score.parts[0] #Melody Input
            hi = score.chordify(addTies=False)  #Harmony Input
            ts = score.getTimeSignatures()[0]   #TimeSig
            ks = mi.analyze('key')              #KeySig

            #Determine how long each bar will be in this work
            start = 0;
            end = len(mi.getElementsByClass('Measure'))
            if mi.getElementsByClass('Measure')[0].paddingLeft > 0:
                start = 0

            mw = [] #Melody for this work
            hw = [] #Harmony for this work

            for i in range(start, end):
                mb = [] #Melody for this bar
                hb = [] #Harmony for this nar
                mx = mi.getElementsByClass('Measure')[i]    #Find the melody bar
                hx = hi.getElementsByClass('Measure')[i]    #Find the harmony bar
                j = mx.offset-mx.paddingLeft    #Make sure to account for the first bar being a pickup
                k = j + ts.barDuration.quarterLength    #Set how long each bar lasts

                #Each sixteenth note, check what note and chord is being played.
                while j < k:
                    mb.append(mx.getElementAtOrBefore(j,[note.Note]))   #Add each note to bar output
                    hb.append(hx.getElementAtOrBefore(j, [chord.Chord]))    #Add each chord to bar output
                    j = j+0.5
                mw.append(mb)   #Add bar to work
                hw.append(hb)   #Add bar to work
            mo.append(mw)   #Add melody for work to list of works
            ho.append(hw)   #Add harmony for work to list of works
            ko.append(ks)   #Add keysid for work to list of works

    return mo, ho, ko
```

Figure 5: A screenshot depicting the segment of code containing the updated version of *streamInput*

In the above figure is the improved version of *streamInput*, rewritten from scratch. Rather than extracting each musical element in sequence, this version of data processing was a little bit more robust. Instead of the previous approach, which outputted each melody and harmony element in sequence, this approach takes into account the durations of each note and chord.

For each bar, this algorithm scans the bar over in eight-note increments, taking note of what note and chord was played most recently. In addition, this results in a consistent amount of elements in each bar. If we consider the same bar of music from before.



```
Melody Scale degrees: [1, 1, 1, 3, 5, 5, 4, 4]
Harmony chords: [I, I, I, I, I, I, IV, IV]
```

Figure 6: A short except of music along with its representation in scale degrees and chords after being processed by the updated *streamInput* function

This feature extraction function is used for both the RNN Method and the LSTM Method, however, the features extracted from the data were processed slightly differently in each method. The Improved LSTM Method made use of a different input function for reasons that will be discussed in its section.

3.3 - The RNN Method: Predicting Harmony Based on Melodic Context

As mentioned before, the goal of the RNN Method was to create an RNN based model that took a section of melody as input, and to produce an appropriate section of harmony to accompany said section of melody. This would result in a simplistic approach of harmonisation to compare methods two and three with later down the line.

There are a number of approaches that exist that may have allowed this project to achieve a solution to this problem, but none seemed to be better suited than seq2seq. Seq2seq is a group of similar approaches used in natural language processing that are utilized to convert one sequence of values into another sequence of values via the use of an RNN.[11] Ordinarily, the seq2seq approach is used frequently within the fields of machine translation and conversational models such as chatbots, but it possesses some properties that are applicable to the problem at hand.

As previously stated in the data processing in section 3.2 of this report, musical data has been formatted into sequences of melodic notes and harmonic chords. This means that the input data would be perfectly suited to this seq2seq approach, as a model could be trained simply using these sequences as training data without too much alteration.

The approach that was ultimately utilized was an approach similar to that found in character-level language models. This follows the premise of using its model to take one sequence of data, in this case the melody, encode it into a machine-readable format, and then use that to predict a different sequence of data, in this case the harmony.

Upon beginning to tackle this problem, however, a problem is immediately obvious. If we trained a neural network on a dataset consisting entirely of music in A major, the network would have difficulty working with other musical keys, despite relationships between notes in those keys being the same as in other keys. Likewise Music21 also keeps track of which octave each note belongs to, and this information could prevent the network from recognizing patterns in the data.

```
def getScaleDegrees(mi, ki):
    print("Scanning scale degrees...")
    degrees = []
    for i in range(0, len(mi)):
        ki = keys[i]

        mx = mi[i]
        if(type(ki) == key.KeySignature):
                ki = ki.asKey()
        for bar in mx:
            mb = []
            for n in bar:
                if (type(n) != note.Note):
                    mb.append(None)
                else:
                    mb.append(ki.getScaleDegreeFromPitch(n.name))
            degrees.append(mb)

    print("Scanning done, building melody vocab...")
    vocabulary = []

    for bar in degrees:
        for n in bar:
            if n not in vocabulary:
                vocabulary.append(n)

    word2idx = {w: idx for (idx, w) in enumerate(vocabulary)}
    idx2word = {idx: w for (idx, w) in enumerate(vocabulary)}

    return degrees, vocabulary, word2idx, idx2word
```

Figure 7: A short excerpt of code depicting the *getScaleDegrees* function

```
def scanChords(hi, ki):
    print("Scanning chords...")
    chords = []
    vocabulary = []

    #For each work
    for i in range(0, len(hi)):

        kw = ki[i]
        hx = hi[i]

        #Make sure KeySig is correct object
        if(type(kw) == key.KeySignature):
            kw = kw.asKey()
        #For each bar in work
        for bar in hx:
            hb = []
            #For each chord in bar, add Roman Numeral to list
            #This simplifies the harmony for machine learnining
            for c in bar:
                if (type(c) != chord.Chord):
                    hb.append(None)
                else:
                    rn = roman.romanNumeralFromChord(c, kw)
                    hb.append(rn.romanNumeral)

            chords.append(hb)
    #Set up vocabulary of available chords and notes
    print("Scanning done, building chord vocab...")
    vocabulary = []

    for bar in chords:
        for c in bar:
            if c not in vocabulary:
                vocabulary.append(c)

    word2idx = {w: idx for (idx, w) in enumerate(vocabulary)}
    idx2word = {idx: w for (idx, w) in enumerate(vocabulary)}

    print("... Done!")
    return chords, vocabulary, word2idx, idx2word
```

Figure 8: A short excerpt of code depicting the *scanChords* function

The solution to this issue is somewhat straightforward, but it shouldn't be understated because it is an important issue to solve. As mentioned before in the music theory section of this report, in section 2.1, the seven triads of any key can be represented using roman numerals. These seven triads, regardless of whatever key they may be part of, will always possess the same relationship to every other chord. By representing data this way, any patterns the model learns to recognise by data in one key will be applicable to data in every key. The same can be done by representing melodic notes as scale degrees rather than as pitches. The functions written to reach this solution are shown in figures above.

In addition to this, the data needed to be encoded into a machine-readable format. According to a 2017 paper: *A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers* [16], the encoding techniques of Sum Coding and Backward Difference Coding are the most accurate encoding techniques available, performing at 95% accuracy in their study. However, it is also noted in the study that One-Hot encoding is the most popular form of encoding, and it possessed a respectable 90% accuracy rating according to their own metrics.

One-Hot encoding is used specifically to make categorical data readable by machine learning models. It is primarily used in classification problems, much like this project. One-Hot allows the input for the model to be represented as vectors. Using these one-hot vectors, data can be fed into the model.

The model constructed for the RNN Method is a two-layer RNN followed by a dense layer. This model utilizes the dropout technique to address overfitting. The dropout technique randomly selects individual neurons during an epoch training and deactivates them for that epoch. In a 2014 publication, it was found that the dropout technique to be extremely effective at reducing overfitting on all neural networks of any size, with the downside of the dropout technique increasing training time.[20]

The loss function selected for this approach was cross entropy loss. Cross entropy loss is well suited to classification models, of which this model is one. With this loss function, the loss increases as the probability of predicting the true label decreases.[21] This results in the model being much more likely to create a correct prediction as it minimizes loss. The optimization function used was adaptive moment estimation, or adam. This is because after attempting to make use of SGD as per the other two methods, the program would crash.

To summarize, this model was trained via a rather simple process. First was separating the harmony bars from melody bars into separate lists. Each of these lists was then processed so that the melody could be expressed in scale degrees, and the harmony could be expressed as roman numerals. The melodic data was fed into the neural network as an input, whereas the harmonic data was used as labels to compare the model's prediction to.

```
Melody scale degrees: [2, 3, 4, 5, 5, 3, 1, 2]
Predicted harmony: ['-VII', 'vi', 'iii', 'V', 'V', 'vi', 'I', '-VII']
```

Figure 9: A short yet jazzy bar fragment of melody represented as a series of scale degrees depicted alongside a predicted bar of harmony to accompany it

As we can see in the figure above, each output of this model is a list, representing a bar of harmony, consisting of chords, each a duration of an eighth note or a quaver, to make up the duration of the bar. Larger criticism of the results of this model will take place in the Results and Evaluation section of this report, but something to immediately note is that each note in a bar will only ever be harmonised by one chord. Every scale degree of 2 is harmonised by '-VII', every instance of chord 3 is harmonised by 'vi'. This shows, as to be expected, a limited amount of understanding for the context of harmony. Ordinarily, we might expect chord choice to differ in order to create more colourful music. For the sake of a more diverse visualization of this output data Figure 10 depicts a representation of the above output in written music. This music can be listened to in the *output_example.mid* file found in the deliverables for this project.

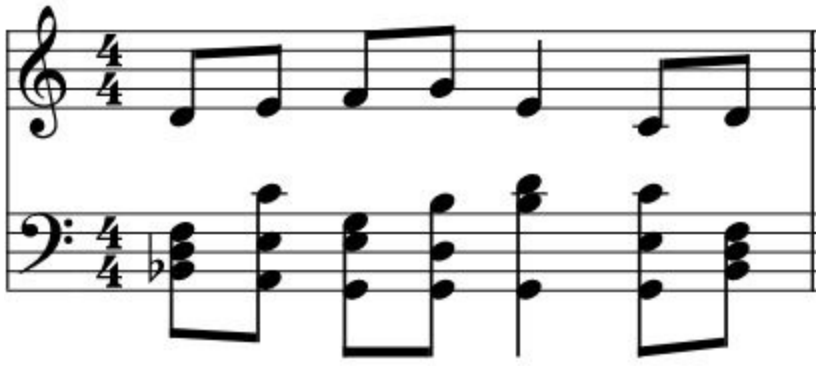


Figure 10: The melody and predicted harmony depicted in Figure 8, but written using standard musical notation

3.4 - The LSTM Method: Predicting Harmony Based on Harmonic Context

The goal of the LSTM Method was to implement LSTM layers into a model and compare the effectiveness of the use of purely harmonic context when training a deep learning model to process music with natural language processing techniques.

LSTM models excel at making use of context in sequential data. In this instance, we wish to ignore any form of melodic data as was used in the last method, and instead purely make use of harmonic data.

This project's investigation into LSTM utilizes two different kinds of word2vec approaches for chord prediction. In the LSTM Method, two distinctive models were created for each of these approaches, both of which utilize some form of a continuous bag-of-words model. In the field of Natural Language Processing, as mentioned in section 2.3, a continuous bag-of-words model attempts to predict a word given the context of surrounding words.

The first model is a simple language modeler that utilizes n-grams, which take n chords prior to the target as an input. These n-grams are associated with a target chord. The second model is that of a birdectional-context language modeler, which was hypothesized to be a much more robust model, is one that takes in n chords prior to the target and n chords post to the target as an input. A sub-goal of the LSTM Method was to compare the effectiveness of these approaches. As might be expected, the primary difference between these two approaches was the way that data was prepared for their input. Both of these types of models are frequently used within the field of natural language processing as language modelers, particularly within the area of sentiment analysis.

To highlight, first language model, that of the n-gram language modeler, utilizes prior context context chord to predict the next chord, where the bidirectional-context language modeler utilizes prior and subsequent chord contexts to make predictions.

Given that two slightly differing models were being produced, we can summarize that the LSTM Method's approach was to create a group of models that predict a target chord based on surrounding context chords, making use of a word2vec approach and utilizing both an embedding layer and LSTM layers.

Upon originally completing the first implementation of this solution, it was observed that the accuracy of the model was extremely low, and the model would keep predicting the most numerous chord that occurred in the context. The issue was bar-long sequences of harmony being used as an input. Consider the following input.

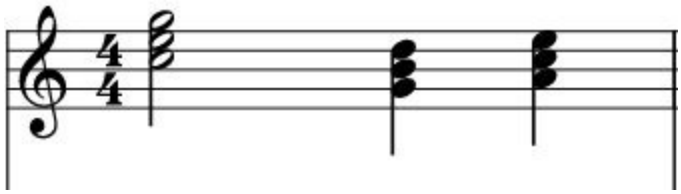


Figure 11: A short excerpt of music consisting of three chords

After feature extraction, this would be represented in the following list: [I, I, I, I, IV, IV, V, V]. This was this issue with the early version of this model. An n-gram such as [I, I] with the target chord [I] doesn't actually tell us a lot about the music. As such, it seemed that the important thing to extract from the music was the change in harmony, rather than the harmony itself. The same passage of music listed in the above figure would instead be represented by the following list: [I, IV, V]. This allows the model to learn a much greater amount about how the harmony functions and resulted in much higher accuracy than the previous method.

```
def scanChordsByNumeral(hi, ki):
    chords = []
    lastChord = None
    for i in range(0, len(hi)):
        hw = []
        kw = ki[i]
        if(type(kw) == key.KeySignature):
            kw = kw.asKey()

        for j in range(0, len(hi[i])):
            hb = hi[i][j]
            if len(hb) > 0:
                for c in hb:
                    if (type(c) == chord.Chord):
                        if lastChord != roman.romanNumeralFromChord(c, kw).romanNumeral:
                            rn = roman.romanNumeralFromChord(c, kw)
                            hw.append(rn.romanNumeral)
                            lastChord = rn.romanNumeral
        chords.append(hw)

    print("Scanning done, building chord vocab...")
    vocabulary = []
    for work in chords:
        for c in work:
            if c not in vocabulary:
                vocabulary.append(c)

    word2idx = {w: idx for (idx, w) in enumerate(vocabulary)}
    idx2word = {idx: w for (idx, w) in enumerate(vocabulary)}

    return chords, vocabulary, word2idx, idx2word
```

Figure 12: An excerpt of code depicting the *scanChordsByNumeral* function utilized by the LSTM method

In the above figure is the function used to prepare data for the LSTM Method's models. Functionally, this function is not dissimilar from the one used in the RNN Method. However, as mentioned above, rather than bar-long sequences, this function returns a

list of every chord as it appears in the harmony in sequence. A new chord is only added to the list when it differs from the previous one. This results in noise entering the network being reduced.

A vocabulary of chords is then constructed and indexed. This assigns a unique integer to each chord, so that this data can be represented in a manner that is a little bit more machine-readable. These are used to create word embeddings in the embedding layer of the model that we will be feeding this into.[14][23]

It is at this point the data is formatted to correctly suit either the n-gram model or the full-context model. Data is converted into lists of context-target pairs. The size of the context is variable, and several different models have been created in order to measure the effect of using different context sizes on models. To give an insight as to how this works, using the n-gram format with a window size of 2 as an example, the sequence [I, IV, V, I, V, I] would result in the following list of context-target pairs: [[[I, IV], V], [[VI, V], I], [[V, I], V], [[I, V], I].

An encoded version of this data split into test and training data via a k-fold function. K-fold cross validation is a technique used to evaluate machine learning models. The k-fold technique involves splitting a dataset into k distinctive groups. One of these groups are then used as a training dataset, whereas the others are used as a testing dataset. This allows a project utilizing a relatively small dataset to find extra mileage out of a limited supply of data. In the case of all models created, each has made use of a k value of 10, resulting in a test dataset 1/10th the size of the training dataset.

The models produced via this method consist of an embedding layer, two LSTM layers, and finally a linear, or dense layer. The input is fed into the embedding layer, which completes the word2vec process by allowing word embeddings to be used in the neural network. The LSTM layers are used to complete the word2vec process by recognizing patterns within the data in order to output a vector representing embeddings in n dimensional space.

The embedding dimensions used differ depending on the model within the group, but the control models use an embedding dimension size of 256. For each type of model, there also exists a version where the embedding dimensions have been set to a smaller size of 128, and another version where the embedding dimensions have been set to a larger size of 512. While the loss function used for this model is the same as the previous model, this model makes use of the SGD optimizer.

To describe the LSTM Method in summary, extracted features were processed to create a sequential list of chord changes. This was used to create a group of models. The differences between these models was the way in which this data was formatted. The group of models was trained by asking it to predict missing target chords from the aforementioned context-target pairs. The result is a group of models that can predict chords solely based on the musical harmonic context, without any melodic context.

3.5 - The Improved LSTM Method: Predicting Harmony Based on a Combination of Melodic and Harmonic Context

The goal of approach three was to find a way to combine the use of melody and harmony when predicting chords. The way this was accomplished was by taking the structure of the group of models in the LSTM Method and finding a way of incorporating melodic data into the context-target pairs utilized by that method. As such, the Improved LSTM Method can be accurately described as a derivative or improvement of the LSTM Method. Very little about the model structure or training differs in this approach from the LSTM Method, but there is a large amount of difference within the data preparation of this approach. As such, if a detail has been failed to be mentioned in this method's description, it can safely be assumed to mimic that of the LSTM Method.

There were several aspects of the LSTM Method that needed to be addressed or scrapped in order to make the Improved LSTM Method's aims possible. In order to express harmonic data and melodic data in the same object, the representation of harmony as roman numerals would need to be scrapped. Instead, an entirely different approach was used. Each chord was instead represented as a string of ordered Pitch Classes.

Pitch classes are a representation of notes as integers, similar to scale degrees. [24] Instead of roman numerals, which represent chords relative to the key they exist in, we can represent chords as strings of pitch classes. For example, we can represent the chord C major as <047>. Using this system, if all music is transposed to the key of C, we have an accurate data point for every chord, that includes information about what melodic note is being played in that chord.

In order to achieve this, the Improved LSTM Method did however make use of a unique feature extraction function compared to methods two and three, as it was required for all pieces of music to be transposed before features were extracted.

Each of these chords, represented as pitch classes, was then allocated into one list, which represents the music as a combination of both melody and harmony. This list was then converted into a vocabulary and indexed via the same process as in the LSTM Method. However, due to the larger amount of possibilities of chord representations, the size of this dictionary is dramatically larger. This results in an incredibly inflated size of the vocabulary in the Improved LSTM Method, as opposed to the one found in the LSTM Method.

This indexed data is then formatted into context-value pairs, and fed into the same model structure as found in the LSTM Method, however, due to this Method making use of both melodic and harmonic context, the Improved LSTM Method results in a much higher accuracy than that found in the LSTM Method.

4 - Results and Evaluation

4.1 - Results Overview

As this project was primarily investigative in intention, a large number of models were produced. These models are able to be categorized by the method which produced them, and within these categories can be subcategorized by the type of model that was produced for each method. Because the LSTM Method and the Improved LSTM Method made use of the word2vec approach during their data preparation stages, a comparison between different word2vec architectures has been included. Within each of these subcategories, models have also been produced by tweaking parameters with these models, to determine the best approach for the situation at hand. Below is a short list containing every model produced by this project. A filename for each model has been provided so that these can be located within the deliverables for this project.

Models Produced Via the RNN Method

- RNN Control *(approach1melodyRNNE256.pt)*

  This model is a basic recurrent neural network that receives a sequence, or bar, of melody as an input, and outputs a sequence, or bar, of harmony. Its hidden dimensions are 256. This model was constructed as a control model to be used in comparison to other models developed using the LSTM and Improved LSTM methods.

- RNN with Smaller Hidden Dimensions (*approach1melodyRNNE128.pt)*

  This model is identical to that of the Control model produced by the RNN method, with the key difference that the hidden dimensions are smaller, instead set to 128. This model was predicted to have a much lower accuracy overall when compared to the RNN control model.

- RNN with Larger Hidden Dimensions (*approach1melodyRNNE512.pt)*

    This model is identical to that of the Control model produced by the RNN method, with the key difference that the hidden dimensions are larger, instead set to 512. This model was expected to have a somewhat higher accuracy overall when compared to the RNN control model, but with a moderate plateau in accuracy after a decent amount of training time.

Models Produced via the The LSTM Method

- N-gram Language Modeler - Control (*approach2NGramC3E256.pt)*

    This model utilizes n-gram context-pairs in its approach to achieve word embeddings. It features an embedding layer, two LSTM layers, the former of which utilizes the dropout technique, and finally a linear layer. This model predicts a target chord represented as a roman numeral from a context of three prior chords.

- Bidirectional-Context Language Modeler Control - (approach2CBOWC2E256.pt)

    This model utilizes context-pairs where the context consists of chords both preceding and following the target chord in its approach to achieve word embeddings. Similar to the n-gram language modeler, It features an embedding layer, two LSTM layers, the former of which utilizes the dropout technique, and finally a linear layer. This model predicts a target chord represented as a roman numeral from a context of three prior chords. Due to the additional direction in which this language modeler is able to consider context, it is predicted that this modeler will result in a higher accuracy than the accuracy found in the ordinary n-gram language modeler.

- Language Modelers with Larger Dimensions (*approach2NGramC3E512.pt* + approach2CBOWC2E512.pt*)*

    These two models are identical to that of the n-gram control model, and the directional-context control model produced by the LSTM method, with the key difference that their embedding dimensions are both larger,

instead set to 512. These models were expected to have a slightly higher accuracy when compared to their control model counterparts. This was expected because with each new embedding dimension, each chord possesses an additional metric to be compared to other chords with, which theoretically should allow for more refined comparison of chords.

- Language Modelers with Smaller Dimensions (*approach2NGramC3E128.pt* + approach2CBOWC2E128.pt*)*

  These two models are identical to that of the n-gram control model, and the directional-context control model produced by the LSTM method, with the key difference that their embedding dimensions are smaller, instead set to 128. These models were expected to have the inverse outcome of the models possessing larger dimensions, whereby decreasing the size of embedding dimensions reduces the metrics available to compare chords with. This is predicted to result in much less accuracy overall when compared with other methods.

- Language Modelers with Smaller Context Windows (approach2NGramC2E256.pt + approach2CBOWC1E256.pt)

  These two models are identical to that of the n-gram control model, and the directional-context control model produced by the LSTM method, with the key difference between them and their control counterparts being that these models possess smaller context sizes. This was expected to negatively impact the overall accuracy of both of these models, as they would possess less information overall to utilize.

- Language Modelers with Smaller Context Windows (approach2NGramC4E256.pt + approach2CBOWC3E256.pt)

  These two models are identical to that of the n-gram control model, and the directional-context control model produced by the LSTM method, with the key difference between them and their control counterparts being that these models possess larger context sizes. This was actually expected to negatively impact the overall accuracy of both of these models, as an increased context size within the context-target pair results

in fewer context-value pairs overall, thus effectively reducing the size of the training dataset.

Models Produced Via the Improved LSTM Method

- N-gram Language Modeler - Control (*approach3NGramC3E256.pt)*

  This model utilizes n-gram context-pairs in its approach to achieve word embeddings. It features an embedding layer, two LSTM layers, the former of which utilizes the dropout technique, and finally a linear layer. The primary difference between this model and the model found in the LSTM method is that this model utilizes melodic data in order to make chord predictions. This model predicts a target chord represented as a roman numeral from a context of three prior chords.

- Bidirectional-Context Language Modeler - Control (*approach3CBOWC2E256.pt)*

  This model utilizes context-pairs consisting of both prior and subsequent chords in its approach to achieve word embeddings. This additional data presented to the model should allow it to make more accurate predictions. However, it is hypothesized that this will require slightly more training to achieve than the simpler n-gram language modeler. It features an embedding layer, two LSTM layers, the former of which utilizes the dropout technique, and finally a linear layer. The primary difference between this model and the model found in the LSTM method is that this model utilizes melodic data in order to make chord predictions. This model predicts a target chord represented as a roman numeral from a context of three prior chords.

4.2 - Comparison of Methods One, Two and Three

The RNN Method, as mentioned previously in the Solution Overview section of this report, was a model built in order to approximate more traditional harmonisation techniques without the use of harmonic context or consideration for chord progression. The RNN Method's function within this investigation is to serve as a control for the other two methods compared against, allowing this project to draw a conclusion about their effectiveness. Below are figures depicting the average loss and the accuracy of the model over 100 epochs of training.

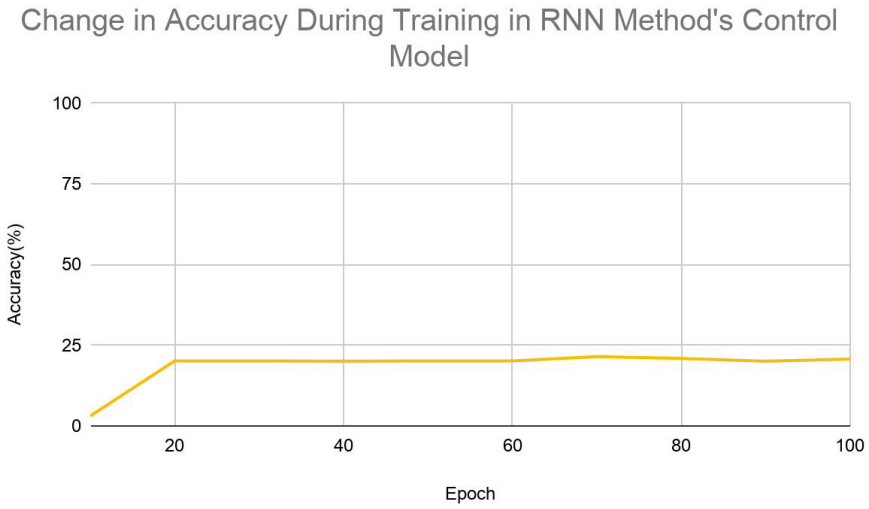


Figure 13: Line Graph depicting change of accuracy over the training duration of the produced RNN model during the RNN method

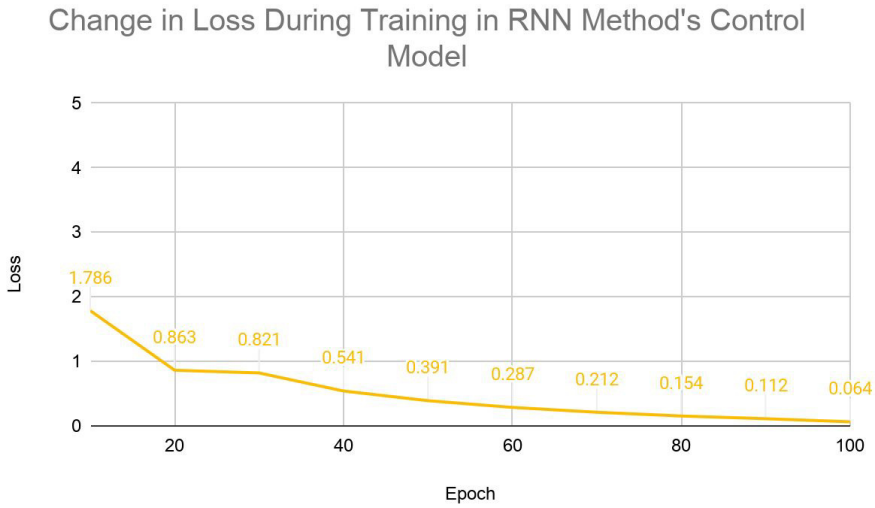


Figure 14: Line Graph depicting change of loss over the training duration of the produced RNN model during the RNN method

The visualization of the data seen above does not paint a particularly promising picture for the RNN Method. As the accuracy plateaus at approximately 20 percent, it seems as though this model isn't tremendously accurate, despite the fact that the model's loss decreased dramatically over the course of the training time. This would be an inaccurate final assessment of this model however, and the reason for this is because the traditional measure of accuracy used in machine learning accounts for the number of identical predictions the model makes to the labels in the dataset. The musical nature of the problem is what complicates these accuracy readings.

```
Melody scale degrees: [1, 1, 3, 1, 3, 5, 5, 5]
Predicted harmony: ['I', 'I', 'vi', 'I', 'vi', 'V', 'V', 'V']
Actual harmony: ['I', 'I', 'vi', 'vi', 'I', 'I', 'I', 'I']
```

Figure 15: A short melody depicted alongside a predicted harmony and its associated ground truth harmony

By comparing some predictions made by the model with their ground truth counterparts, a musical eye might point out that while a number of chord choices for this bar might be a bit abrupt, they are entirely functional within the musical context in which they exist. For example, the scale degree 3 is able to be used in many chords. An appropriate chord choice for this scale degree might be chord iii, chord I or chord vi. As such, it is hard to describe all of this model's chord choices inaccurate, as it wouldn't be strictly incorrect to use the majority of these chord choices, even though they differ from the ground truth. As such, an alternative formula for accuracy has been used in the following graph. In this formula, a prediction has been considered accurate if a scale degree belongs to the predicted chord.

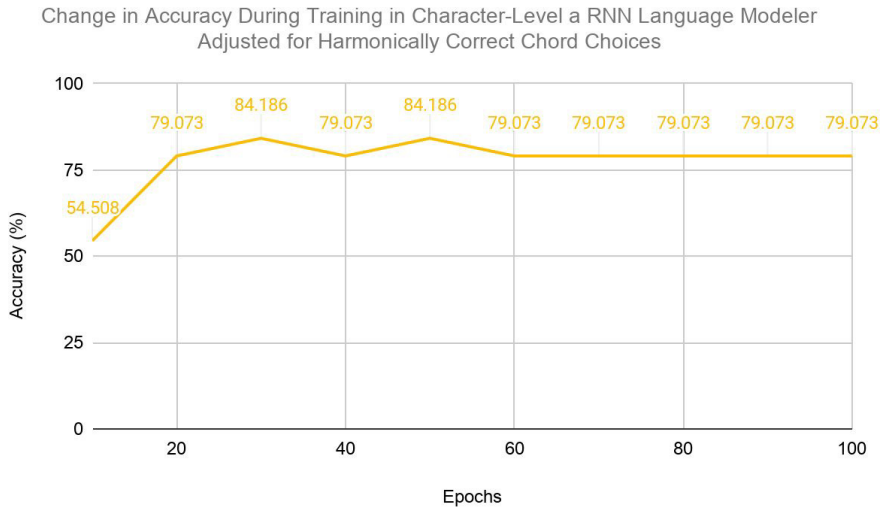


Figure 16: Line Graph depicting adjusted change of accuracy over the training duration of the produced RNN model during the RNN method

Upon observation of this new data, we can see that a reasonably large percentage of the RNNs predictions are musically valid, if not identically accurate to the associated ground truth. This figure depicts a much more reasonable accuracy for this Method. However, it is still clearly visible from this visualization that the graph begins to plateau at about epoch 30, and so there is probably not going to be a large increase in accuracy without a change in the way data is processed before being fed into this network.

Overall, as predicted, this method seems to give us usable harmonic choices even if they are reasonably fixed. As expected, the lack of chord context is something that can really hurt the chord choice, however.

The LSTM Method sought to contrast itself with the RNN Method by strictly making use of harmonic context. It was anticipated that this method would not suffice on its own to be an effective means of harmonisation, and that the combination of melodic and harmonic data would be required in order to make harmonic data effective. However, to some surprise a reasonable level of accuracy was able to be achieved from this approach.

The LSTM Method made use of two different types of model with differing architectures. The results of the comparison between different model architectures is discussed later in section 4.3, however. For the sake of this comparison, this section will be comparing the LSTM Method's continuous bag-of-words control model, as that was the most accurate out of the control models produced by the LSTM Method.

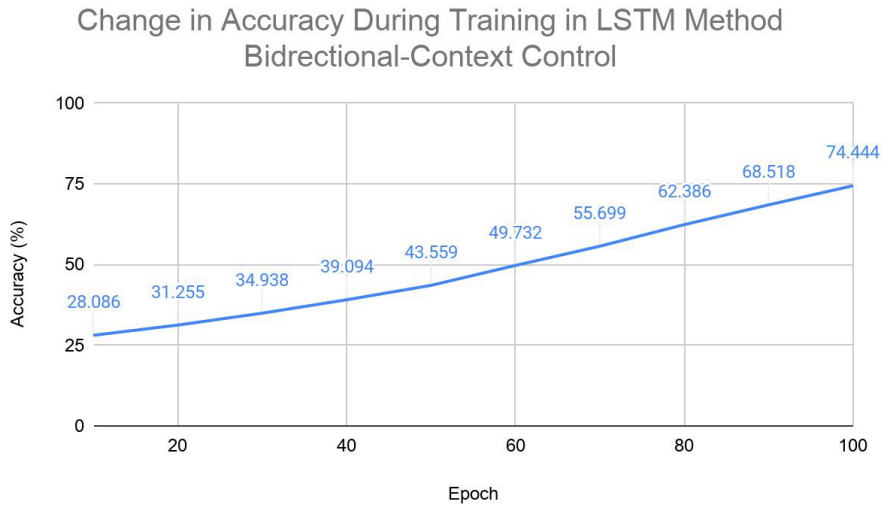


Figure 17: Line Graph depicting change of accuracy over the training duration of the produced LSTM word2vec model during the LSTM Method

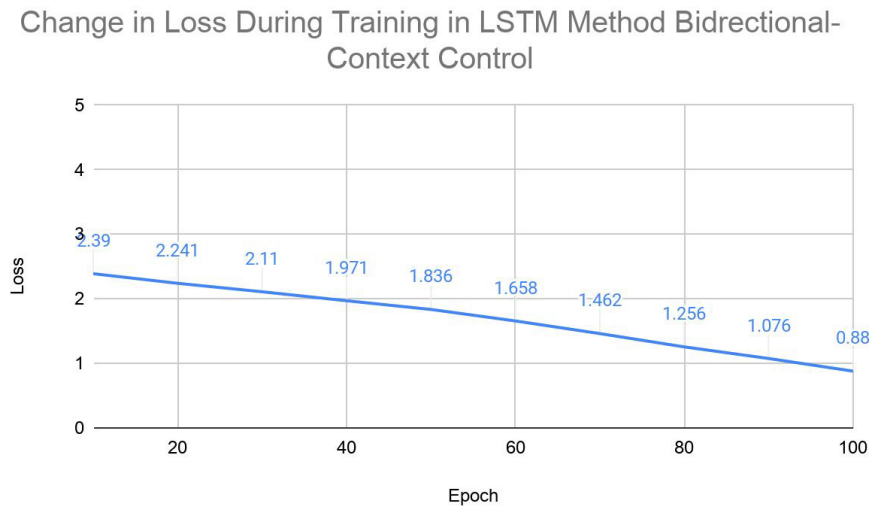


Figure 18: Line Graph depicting change of loss over the training duration of the produced LSTM word2vec model during the LSTM Method

Over the course of 100 epochs of training, the LSTM Method's model reached a respectable 74.4% accuracy, whilst minimizing loss at a steady rate. We can observe the accuracy engaged in a steady upward trend, which by the end of the 100 epochs of training, do not show a clear sign of plateauing, which indicates that with additional training epochs, this model still has the opportunity to increase in accuracy.

This result is slightly surprising, as it was not hypothesized for this method to be as accurate as it ended up being. What makes these results interesting is the implication

that chord patterns without any melodic assistance have some form of semantic meaning that can be understood on their own.

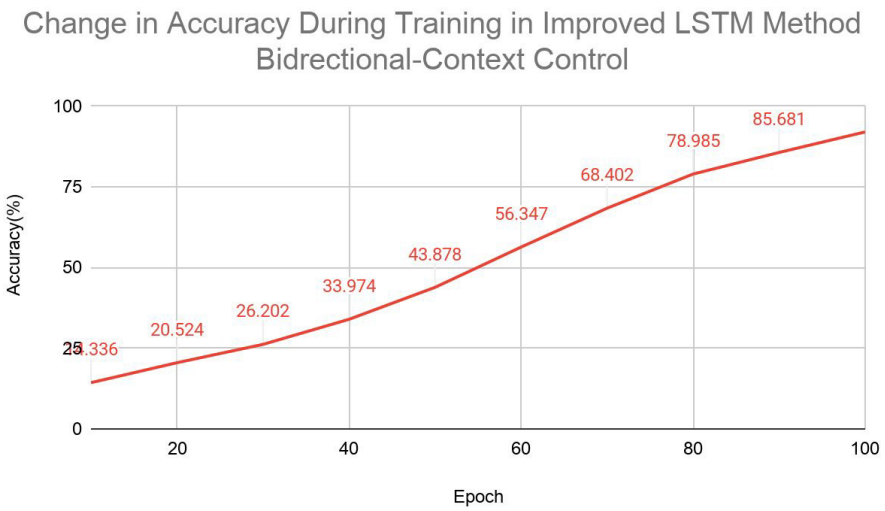


Figure 19: Line Graph depicting change of accuracy over the training duration of the produced LSTM word2vec model during the Improved LSTM Method

Figure 20: Line Graph depicting change of accuracy over the training duration of the produced LSTM word2vec model during the Improved LSTM Method

The approach of using Pitch Class representation to allow for voice leading seems to have worked well for this model, as it has the highest accuracy of all produced models. We can see a strong upward trend, even continuing at later epochs, signifying that with

additional training, the accuracy of the model may yet increase. The loss steadily decreases over each training epoch.

However, starting accuracy seems much lower than that of the previous LSTM-based method, perhaps signifying that the increased vocabulary-size means that the model has a harder time accurately predicting a target chord without much training.

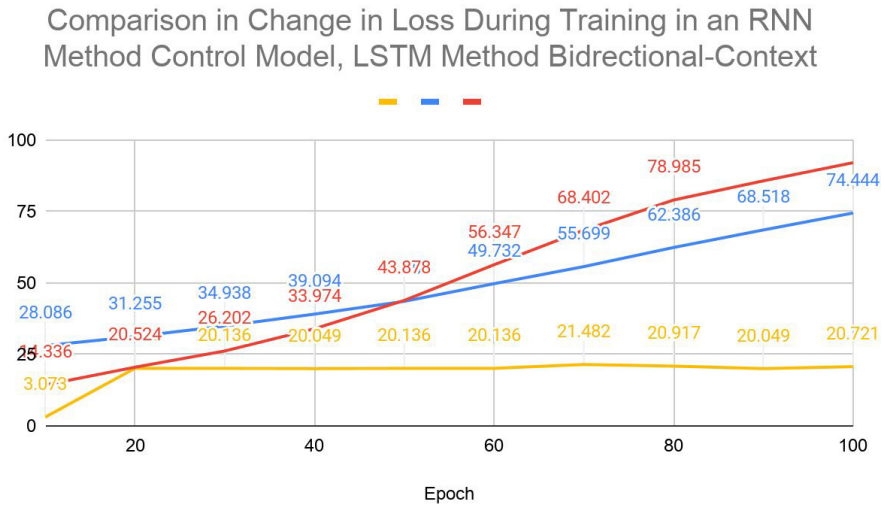


Figure 21: Line Graph depicting a comparison in change of accuracy over the training duration between the control model for each method

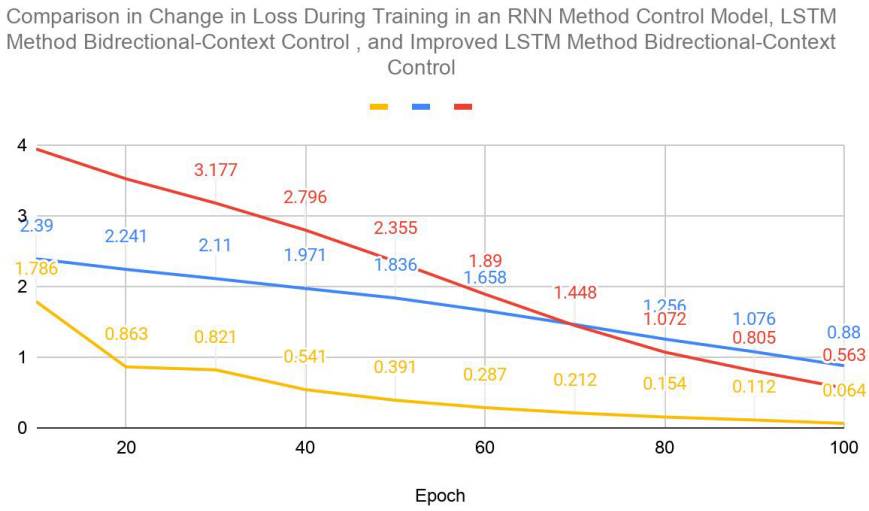


Figure 22: Line Graph depicting a comparison in change of accuracy over the training duration between the control model for each method

When each of the three approaches are overlaid onto the same graph, we can very clearly see that only after about 50 epochs does the additional data available to the

Improved LSTM Method's improved LSTM model begin to outweigh its increased complexity and vocabulary size. Before this point the LSTM Method's simplicity allows it to maintain a higher level of accuracy than its counterparts.

While the RNN Method's accuracy maintains a low plateau for most of the training duration, its loss increases dramatically over time. We can see the influence that this has on the Improved LSTM Method's loss when compared to both the RNN Method and the LSTM Method. The smooth downwards curve allows for the Improved LSTM Method and the RNN Method to maintain a much greater rate of loss than the LSTM Method, and this is likely because of the melodic context being utilized by both of these models.

In addition, the data suggests that the combination of melodic context and harmonic context is indeed the best strategy to aim for when processing music via machine learning. This highlights the utility of LSTM neural networks as an extremely versatile tool for this area, and it is the belief of this investigation that LSTM neural networks possess far more use in the field of music processing than their more standard RNN counterparts.

However, the generally positive results of the models produced by this investigation seem to suggest that natural language processing techniques do have a place within music processing via deep learning.

4.3 - Comparison of word2vec Architectures in Methods Two and Three

The LSTM Method and the Improved LSTM Method made use of a model architecture based upon the word2vec approach. Word2vec, when utilized within its more familiar application of natural language processing, operates by receiving an input of text. This text would ordinarily be referred to as a corpus. This corpus is then systematically broken down, or tokenized into constituent parts. Within the aforementioned context of natural language processing, these constituent parts are usually words, but can also be characters or morphemes. The output of this approach is a series of vectors, representing each of these individual words or tokens.

An explanation of the natural language processing application of this approach has been used to provide an analogue to the musical application in this project, as there is little difference in the techniques used between these two applications, and for those not particularly well versed in music theory, this analogue might provide better understanding of the approach used in this project when read in conjunction with the short description of musical theory concepts in section 2.1 of this report.

In order to experiment with how effectively word2vec could be utilized, this project produced two contrasting models in both methods two and three utilizing the word2vec approach in a different way. In each method, both a simple n-gram model was produced, alongside a counterpart continuous-bag-of-words model. The primary differences between these models are the context chords that are required for prediction. The produced n-gram models require the context of n prior chords to predict a target chord, whereas the produced continuous-bag-of-words models require the context of n prior chords and n post chords.

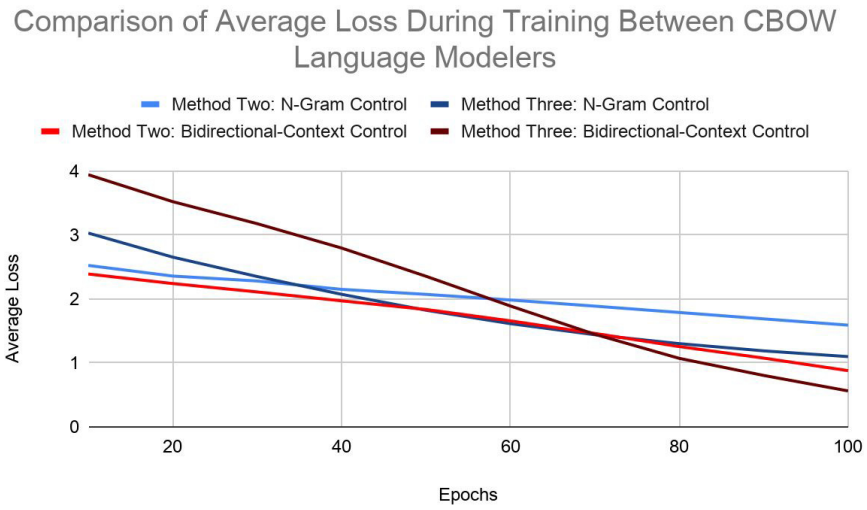


Figure 23: Line Graph depicting a comparison in change of loss over the training duration between the n-gram models produced in the LSTM Method and Improved LSTM Method, and the bidirectional-context models produced in the LSTM Method and Improved LSTM Method

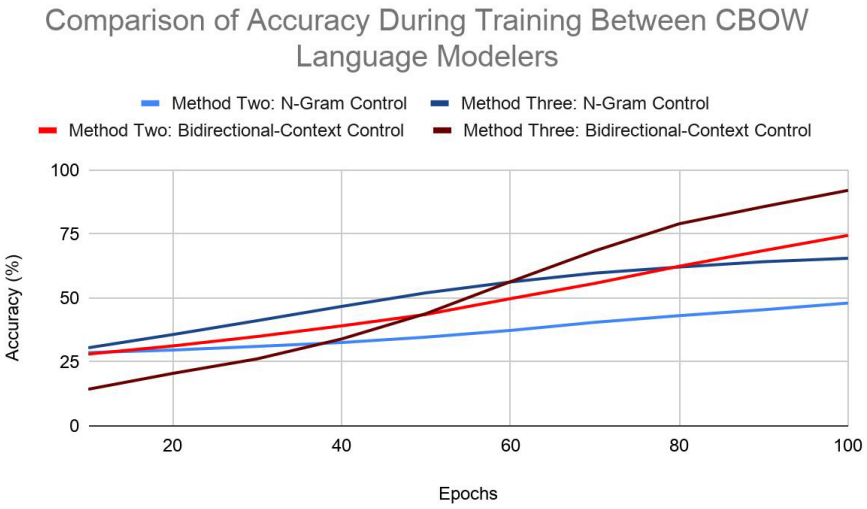


Figure 24: Line Graph depicting a comparison in change of accuracy over the training duration between the n-gram models produced in the LSTM Method and Improved LSTM Method, and the bidirectional-context models produced in the LSTM Method and Improved LSTM Method

Before experimentation, the assumed outcome was that the models utilizing the continuous-bag-of-words structure would be more effective. As can be seen in the visualization above, this does indeed seem to be the case. After completing one hundred epochs of training, the accuracy of both models utilizing a bidirectional-context language model surpassed the accuracy of their n-gram language model counterparts, and also outperformed the n-gram models in terms of minimizing loss. This is very likely due to the fact that the bidirectional-context language models are making use of a larger

context-size in total, despite the length of each window being shorter than the context-size of the n-gram models. The increase in context-size allows for more data to be utilized in prediction, which should increase accuracy. In addition, the bidirectional-context language model has an innate advantage over the n-gram language models due to its ability to utilize chords following the target chord. In terms of a musical conceptualization of the problem, this would allow the model to develop a better understanding of the semantic meaning of the harmony, allowing for concepts such as voice leading or chord progressions to potentially be understood better.

However, this doesn't mean that the bidirectional-context language models are strictly better in every regard. When limited training time is available, such as between epoch 0 and epoch 30, the n-gram models outperform the bidirectional-context language models in terms of accuracy. There are also use-cases where a bidirectional-context language model might not be suitable. A more sophisticated neural network tasked with the composition of music may not have access to harmony that follows a target chord due to a lack of music existing past the target chord, for instance. In such cases of such a model being tasked with composition, the use of a more simple n-gram model could be used to generate a basic structure of music that could then be refined via the use of a bidirectional-context language model which ensures each chord in the harmony is correct.

One prominent word2vec architecture that has not been measured is that of the continuous skip-gram architecture. A skip-gram model, when given an input of a word, will try to predict several context words. This is functionally the inverse of the continuous bag-of-words models that have been produced. The skip-gram architecture has not been investigated due to the fact that while this type of architecture might possess useful traits for musical composition, it is harder to find as significant a use for this type of model in harmonisation when compared to the continuous bag-of-words architecture, and the time-constraints of this have limited this project's scope in terms of the number of architectures that can be investigated. This will be discussed further in section 6 of this report.

4.3 - Comparative Effects of Changing Embedding Dimensions on Models

As each model in the LSTM Method was produced, a number of tweaks were made in an attempt to determine what settings were most effective during model training. This section of the report seeks to document a comparison of different embedding dimensions, or E, during the word2vec process. Three settings were used. First, a control setting where E = 256, a setting with larger embedding dimensions where = 512, and a setting with smaller embedding dimensions where E = 128. In all other models, E = 256 was used. It was this project's hypothesis that generally, larger embedding dimensions would allow for a greater accuracy, with diminishing returns as the embedding dimensions increased

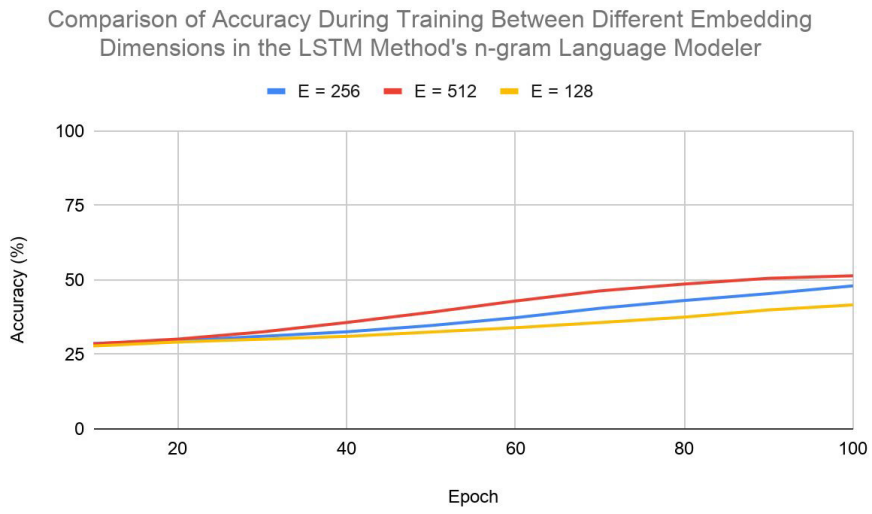


Figure 25: Line Graph depicting a comparison in change of accuracy over the training duration between the n-grams produced during the LSTM Method, with differing embedding dimensions

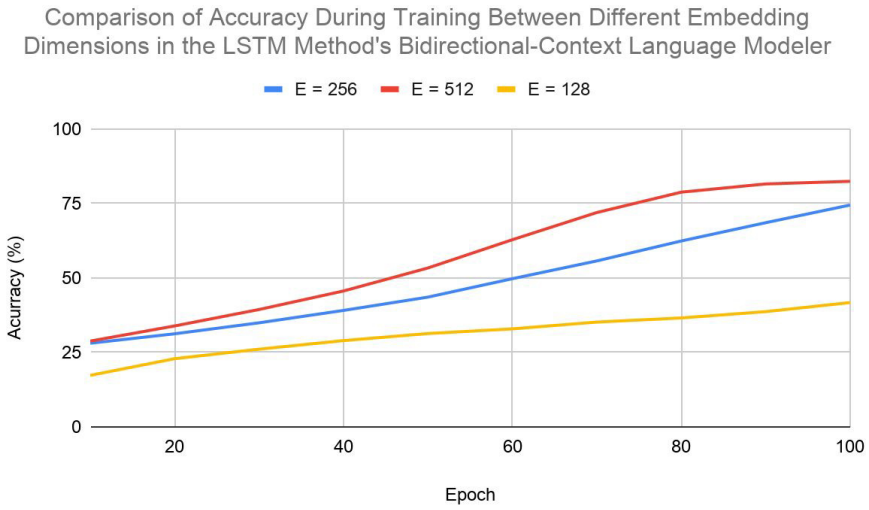


Figure 26: Line Graph depicting a comparison in change of accuracy over the training duration between the bidirectional-context models produced during the LSTM Method, with differing embedding dimensions

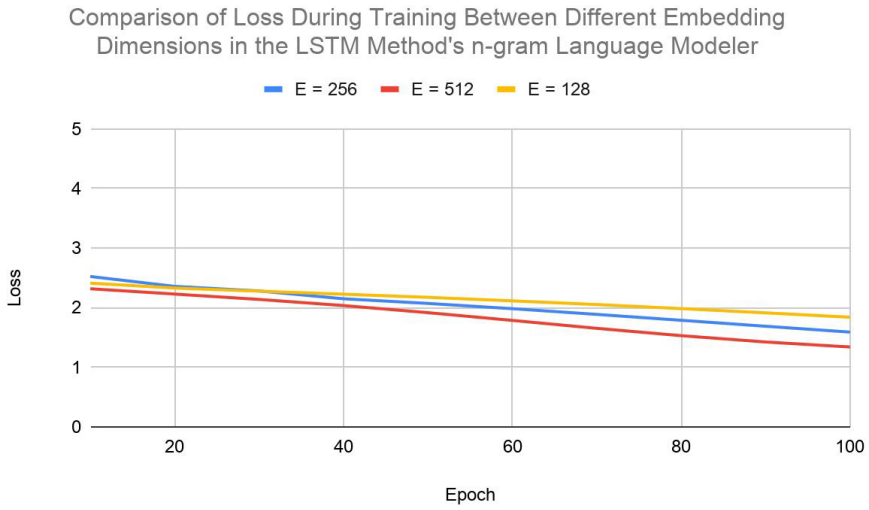


Figure 27: Line Graph depicting a comparison in change of loss over the training duration between the n-gram models produced during the LSTM Method, with differing embedding dimensions

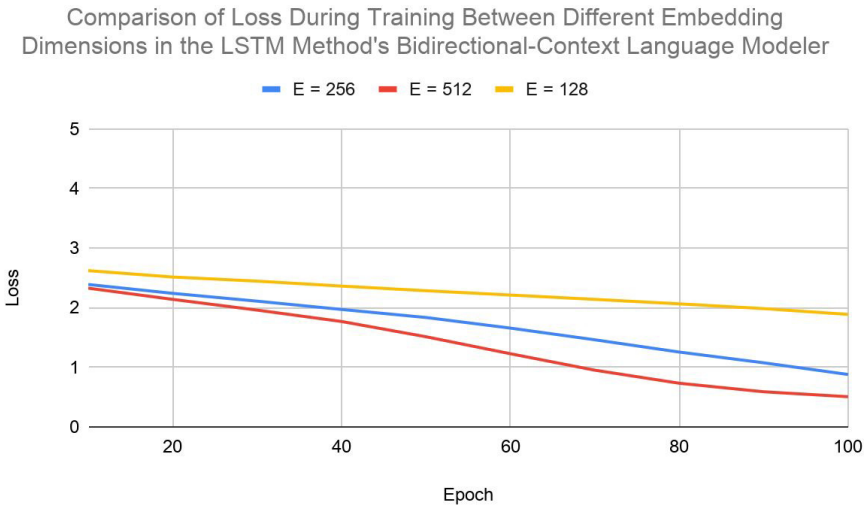


Figure 28: Line Graph depicting a comparison in change of loss over the training duration between the bidirectional-context models produced during the LSTM Method, with differing embedding dimensions

Considering the data in the four figures above, we observe that in all cases, the models with E = 128 dimensions have a slower rate of increase in accuracy per epoch when compared to the other models of E = 512 and E = 256. In addition, this model possesses a slower rate of decrease in loss per epoch. This data would suggest that generally a use of smaller embedding dimensions results in a model performing poorly when compared to the control model of E = 256. In addition, these models appear to plateau in terms of accuracy near the end of the training cycle, which suggests that even with additional training, there wouldn't be much increase in accuracy. This suggests that E = 128 is limited in terms of maximum potential accuracy when compared to its counterparts with more embedding dimensions.

By contrast, we can see that a model with E = 512 tends to possess a larger rate of increase in accuracy per epoch, and a faster rate of decrease in loss per epoch when compared to the other models. This on the surface would imply that larger embedding dimensions are strictly better than the control model. However, towards the end of the training cycle, it can be observed that the rate of increase in accuracy starts to decline, where comparatively the control model of E = 256 does not show any slowing in rate of increase of accuracy. This suggests that over additional training epochs, larger embedding dimensions might be a limiting factor towards total accuracy of model.

It can potentially be assumed from this data that there exists a 'sweet spot' of embedding dimensions to aim for. Using too low a setting of embedding dimensions can

hamper a model's rate increase of accuracy, and decrease of loss over a training period, but too large a setting of embedding dimensions can limit a model's potential maximum accuracy.

4.4 - Comparative Effects of Extending Window Size on Continuous Bag-Of-Words Models

Another interesting point of optimization is that of the context or window size used in the continuous bag-of-words models produced during the LSTM Method. As mentioned before, the window size refers to the size of context within context-target pairs. Within the context of the n-gram language modeler, a window size of 2 might look like [[0,0],1], whereas a window size of 4 might look like [[0,0,1,2],3].

It was this project's hypothesis that as was observed in the experiment involving embedding dimensions, too small of a context size would hamper the learning rate of the model. However, too large of a context size would limit the learning rate much more harshly due to the fact that a lot of the likelihood of encountering a set of context chords would logically decrease as the length of that set increases.

Something to consider is that in the n-gram language models, a context size of 2 will construct a context of two chords prior to the target, whereas in the bidirectional-context language model, a context size of 2 will construct a context of two chords prior to the target and two chords subsequent to the target, resulting in a total of four chords utilized. Because of this, smaller context sizes for the bidirectional-context language model have been used to allow for a fairer comparison.

Three settings were used for each type of model. In the n-gram models, a control setting of C = 3 was used, along with a larger setting of C = 4, and a smaller setting of C = 2. In the bidirectional-context language model a control setting of C = 2 was used, along with a larger setting of C = 3 and a smaller setting of C = 1.

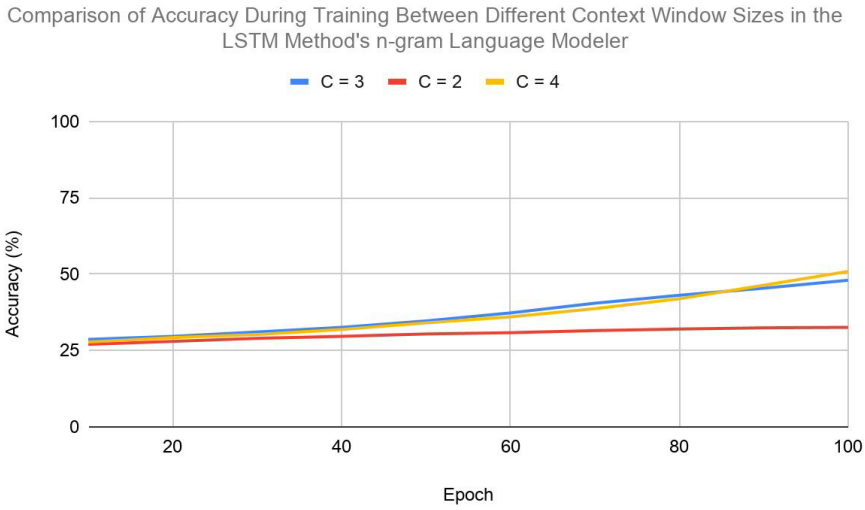


Figure 29: Line Graph depicting a comparison in change of accuracy over the training duration between the n-gram models produced during the LSTM Method, with differing context sizes

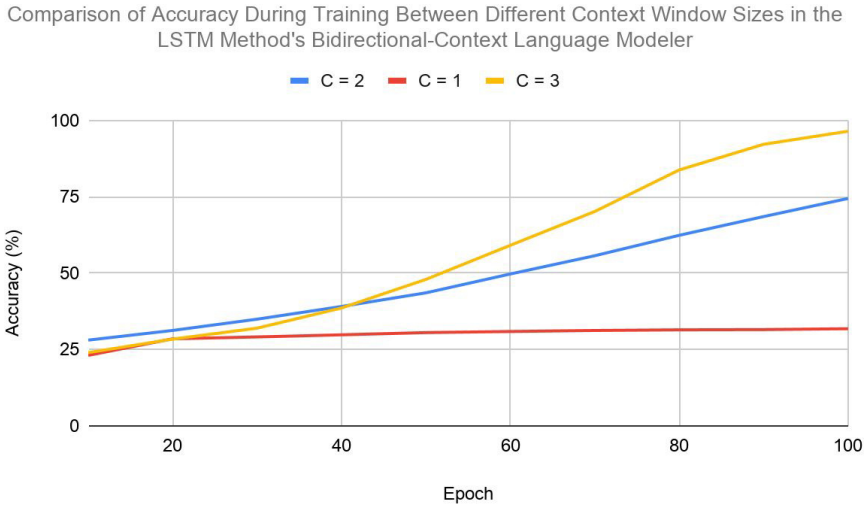


Figure 30: Line Graph depicting a comparison in change of accuracy over the training duration between the bidirectional-context models produced during the LSTM Method, with differing context sizes

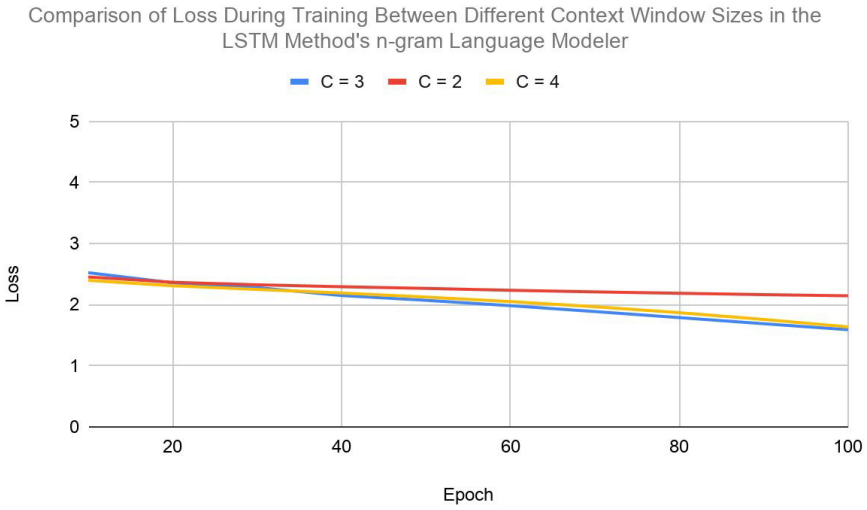


Figure 31: Line Graph depicting a comparison in change of loss over the training duration between the n-gram models produced during the LSTM Method, with differing context sizes

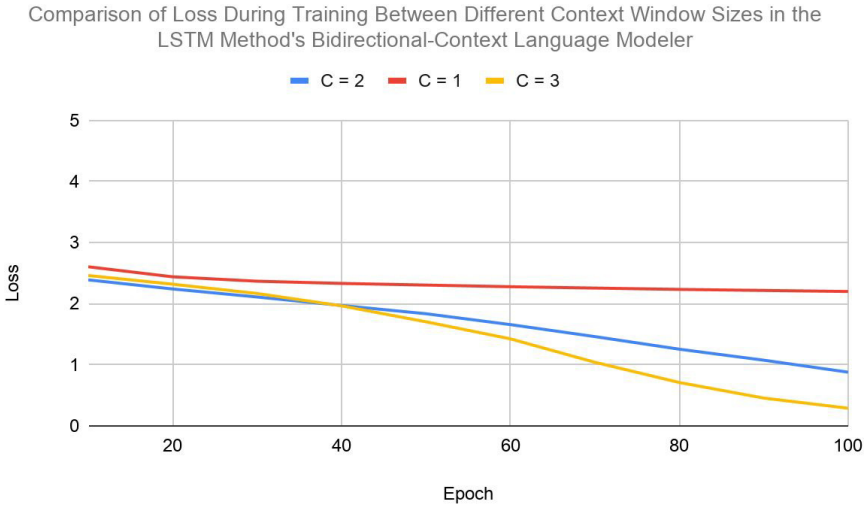


Figure 32: Line Graph depicting a comparison in change of loss over the training duration between the bidirectional-context models produced during the LSTM Method, with differing context sizes

The above results did not prove the hypothesis proposed, but the data suggests some interesting trends nonetheless. As expected, a smaller context size of C = 2 in the n-gram model, and C = 1 in the bidirectional-context language model resulted in a far reduced rate of increase in accuracy per epoch in both models, and a far reduced rate of decrease in loss per epoch in both models. Like is likely due to the reasons hypothesized earlier. The data suggest that there is limited practical usage for such small context sizes, and generally larger ones are to be preferable for a use-case such as this one.

The interesting section of the data would be that of the larger context sizes of C = 4 in the n-gram model and C = 3 in the bidirectional-context language model. For the first fifty epochs, the rate of increase in accuracy and rate of decrease in loss appear to be approximately equivalent to the rates observed within the control models of C = 3 and C = 2. However, the latter fifty epochs seem to constitute a rapid increase rate of increase in accuracy and a slightly increased rate of decrease in loss. This boosts the bidirectional-context language model to a near 99% accuracy by the end of the training epoch.

It seems that the above data would suggest that larger context sizes are strictly better than smaller ones, but it seems doubtful that this is the case and further experimentation will need to be done in order to confirm this.

5 - Conclusions

The inspiration of this project was to consider the ideas of Leonard Bernstein's thesis. This project aimed to investigate the validity of conceptualizing music via the lens of linguistics by using a deep learning model utilizing techniques from the field of natural language processing to harmonise music.

The least effective natural language processing technique was that of the RNN Method, which possessed a maximum accuracy result of 20.7%. This method utilized a character-level language model to predict sequences of harmony for given sequences of melody. It is expected that this low accuracy was due to a common problem mentioned in section 2.2 of this report, that RNNs possess a limited capability for retaining data over longer sequences.

The Improved LSTM Method, which utilized an LSTM neural network and word2vec word embeddings achieved a greater level of accuracy, making use of both melodic and harmonic contexts. The LSTM Method even possessed a decent level of accuracy when simply utilizing harmonic context to make predictions. Given the observations made from the data collected, the data suggests that LSTM neural networks provide a far superior ability to allow for the processing of music with some consideration for musical structure.

In a comparison of data preparation approaches, the bidirectional-context language modeler was found to be more effective after a full cycle of training than its n-gram language modeler counterparts. Both approaches however, seemed to demonstrate competency for processing of musical data in the context of machine learning.

It is the conclusion of this project that to a reasonable degree the use of deep learning natural language processing techniques are applicable to the field of music with a level of effectiveness that seems to support the thesis put forward by Bernstein.

Within the deliverables for this project exist a folder of every model produced, as well as the .ipynb file used to create each of these models.

6 - Future Work

There have been a number of points mentioned throughout this report that indicate further work in this field would be worth investigating. Primarily it would seem that a wider array of techniques also found in the field of natural language processing would need to be investigated. Some of these techniques were briefly mentioned in this report but were not investigated in the study due to time and scope restraints.

The use of sentiment analysis techniques would be an incredibly promising area to explore. Considering that music as an art form is used to convey emotional sentiment in order to elicit an affective response from a listener, an understanding as to how machine learning can process such emotional sentiments would be invaluable in aiding machine-driven composition.

Within areas that were investigated during this report, there could potentially be more depth reached with additional study. One could foresee value in documenting the effectiveness of other innovations on basic recurrent neural networks like long short-term memory networks, such as gated recurrent units, in regard to their use in the area of music processing. In addition, more work could be done on word2vec's potential. One type of word2vec architecture that was mentioned in the wordvec comparison that was not investigated was of a skip-gram model. Such a model would be able to predict context from a given chord. Potential value could be envisioned in a generative model.

Which leads into what is perhaps the most imaginatively appealing area of further research, the development of a generative model for the creation of music using the techniques and approaches documented within this report. Developing such a model would allow the developer to build upon techniques discussed in the report and allow for these techniques to be demonstrated in a more practical setting.

Such a project would be able to make use of the groundwork laid out here. For proof of concept, a short midi file using the output of a model produced via the RNN method has been provided in the deliverables to demonstrate the possibility of this.

7 - Reflection and Learning

Upon reflection of this project, there are a small number of changes that would be made if it were to be repeated, as I feel like a handful of differences could be implemented in order to expedite progress on development, and to make general management of this progress much simpler.

Perhaps the largest road bump that repeatedly presented itself was the lack of experience that I had with machine learning at the start of this project. The amount of necessary reading in order to begin making progress with the project would decrease dramatically if this project was undertaken by an individual with more starting experience. While I began to understand enough about its usage after an amount of practice with PyTorch to begin developing models, I believe in retrospect Keras would have been a better machine learning library to choose for a beginner. Observing a peer who incidentally was a fellow machine-learning novice develop models quickly and with relative ease began causing doubt with PyTorch as a library. Additional time saved on learning is more time that can be spent on experimentation.

The lack of experience in this area was a factor hanging over me from day one and found itself to be incredibly hampering throughout the course of the project, as whenever progress was made, such as with the *streamInput* functions mentioned in section 3.2, something would inevitably break, and this started to get disheartening. Through perseverance knowledge was gained, and solutions to problems were found, but asking for additional help where necessary would definitely be advice I would give to myself before starting this project again.

More time spent on experimentation is a theme echoed in the next change I would make to this project: investigating more techniques for musical harmonisation in total. I believe if time was saved by spending slightly less time conducting background research, perhaps a few extra techniques could have been investigated, leading to a slightly more well-rounded analysis of techniques. If this project wants to truly maintain Leonard Bernstein's thesis that language can be understood through the lens of linguistics, further confirmation may be necessary than this report was able to provide.

One large improvement that could be made to the project is the use of additional datasets. The project since its inception, right through until the final month intended to make use of both a Johann Sebastian Bach dataset and a dataset of Jazz standards.

However, a free-to-use Jazz standard repository stored in MusicXML format was almost impossible to find. Several partial datasets were found, but each of them had many problems with noise that stemmed from issues such as lack of quantization. More time in future needs to be spent on finding an optimal dataset to use.

8 - Bibliography

[1] Eck D., Schmidhuber J. (2002) Learning the Long-Term Structure of the Blues. In: Dorronsoro J.R. (eds) Artificial Neural Networks — ICANN 2002. ICANN 2002. Lecture Notes in Computer Science, vol 2415. Springer, Berlin, Heidelberg

[2] Migdal, P. and Jakubanis, R., 2020. Keras Or Pytorch As Your First Deep Learning Framework - Deepsense.Ai. [online] deepsense.ai. Available at: <https://deepsense.ai/keras-or-pytorch/> [Accessed 12 May 2020].

[3] Jäncke, L., 2012. The relationship between music and language. Frontiers in psychology, 3, p.123.

[4] Nabi, J., 2020. Machine Learning — Text Classification, Language Modelling Using Fast.Ai. [online] Towards Data Science. Available at: <https://towardsdatascience.com/machine-learning-text-classification-language-modelling-using-fast-ai-b1b334f2872d?gi=df13419227cf> [Accessed 12 May 2020].

[5] Magenta. 2020. Coconet: The ML Model Behind Today'S Bach Doodle. [online] Available at: <https://magenta.tensorflow.org/coconet> [Accessed 12 May 2020].

[6] Faria, W. (2018). MIDI Music Data Extraction using Music21 and Word2Vec on Kaggle. [online] Towards Data Science. Available at: https://towardsdatascience.com/midi-music-data-extraction-using-music21-and-word2vec-on-kaggle-cb383261cd4e [Accessed 25 Jan. 2020].

[7] Skúli, S. (2017). How to Generate Music using a LSTM Neural Network in Keras. [online] Towards Data Science. Available at: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5 [Accessed 29 Jan. 2020].

[8] Huang, C.Z.A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., Dai, A.M., Hoffman, M.D., Dinculescu, M. and Eck, D., 2018. Music transformer: Generating music with long-term structure.

[9] Roberts, A., Huang, A., Hawthorne, C., Howcroft, J., Wexler, J., Hong, L. and Dinculescu, M., 2019. Approachable music composition with machine learning at scale.

[10] Bernstein, L. (1973). The Unanswered Question: Lecture 2, "Musical Syntax".

[11] Bernstein, L. (1973). The Unanswered Question: Lecture 3, "Musical Semantics".

[12] Wadhwa, M., 2020. *Seq2seq Model In Machine Learning - Geeksforgeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/seq2seq-model-in-machine-learning/> [Accessed 30 May 2020].

[13] Ruder, S., 2020. *An Overview Of Gradient Descent Optimization Algorithms*. [online] ruder.io. Available at: <https://ruder.io/optimizing-gradient-descent/> [Accessed 31 May 2020].

[14] Nicholson, C., 2020. *A Beginner's Guide To Word2vec And Neural Word Embeddings*. - *Pathmind*. [online] Available at: <https://pathmind.com/wiki/word2vec> [Accessed 31 May 2020].

[15] Sarkar, D., 2018. *Implementing Deep Learning Methods And Feature Engineering For Text Data: The Skip-Gram Model - Kdnuggets*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-skip-gram.html> [Accessed 1 June 2020].

[16] Potdar, Kedar & Pardawala, Taher & Pai, Chinmay. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. International Journal of Computer Applications. 175. 7-9. 10.5120/ijca2017915495.

[17] Vasudev, R., 2017. *What Is One Hot Encoding? Why And When Do You Have To Use It?* | *Hacker Noon*. [online] Hackernoon.com. Available at: <https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f> [Accessed 1 June 2020].

[18] FloydHub Blog. 2019. *Beginner'S Guide On Recurrent Neural Networks With Pytorch*. [online] Available at: <https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/> [Accessed 30 May 2020].

[19] Lopez Yse, D., 2019. *Your Guide To Natural Language Processing (NLP)*. [online] Towards Data Science. Available at:

<https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1> [Accessed 2 June 2020].

[20] Srivastava, N., Hinton, G., Sutskever, I. and Salakhutdinov, R., 2014. *Dropout: A Simple Way To Prevent Neural Networks From overfitting*. [online] Jmlr.org. Available at: <http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf> [Accessed 2 June 2020].

[21] Parmar, R., 2018. *Common Loss Functions In Machine Learning*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23> [Accessed 3 June 2020].

[22] Zhang, L., Wang, S. and Liu, B., 2018. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), p.e1253.

[23] Karani, D., 2018. *Introduction To Word Embedding And Word2vec*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa> [Accessed 3 June 2020].

[24] Open Music Theory. 2020. *Pitch (Class) – Open Music Theory*. [online] Available at: <http://openmusictheory.com/pitch(Class).html> [Accessed 4 June 2020].

[25] Keller, H., 1978. *Was Bach A Mathematician ?*. [online] Harpsichord.org.uk. Available at: <http://www.harpsichord.org.uk/wp-content/uploads/2015/04/bachmath.pdf> [Accessed 4 June 2020].

[26] Phi, M., 2018. *Illustrated Guide To LSTM'S And GRU'S: A Step By Step Explanation*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21> [Accessed 4 June 2020].

[27] Web.mit.edu. 2020. *Music21: A Toolkit For Computer-Aided Musicology*. [online] Available at: <http://web.mit.edu/music21/> [Accessed 5 June 2020].