Generation of Music Using Long Short-Term Memory Recurrent Neural networks

BACHELOR OF SCIENCE (HONOURS) IN SOFTWARE DEVELOPMENT

Written by Uamhan Mac Fhearghusa.

Supervisor: Martin Hynes.

Submitted Date 26/04/2019

Table of contents

Introduction

In This document we will discuss the topic of the generation of music through long short-term memory recurrent neural networks. In short, we explore the use of using data science tools, to create a new musical composition. The creation of new music has typically always been a creative process taken on by humans. But as anyone who has studied music theory will know there are set rules on how music should be created. To break those rules usually produces unpleasing results. To break these constraints and get pleasing results, usually requires virtuoso level of skill/knowledge or luck. This said many musicians do not know these defined rules yet follow them unbeknownst to themselves as it just sounded "good" to them. So, if we have a set of rules that defines how a piece of music should be written, we should be able to write a program that can take advantage of these rules to produce music compositions. For the purpose of this project the use of a neural network was the means decided upon to approach this task.

A neural network was or more specifically a long short-term recurrent neural network. Was chosen for this project. The use of a neural network was chosen over simply hardcoding the rules of music theory and randomly selecting notes that fall within these rules as it was felt that this method would produce compositions that are more distinctly "robotic" in nature. The use of a neural network allows us to be a little looser with the rules. With the goal of producing something that can be mistaken as a human composition.

In the following pages of this dissertation, some music theory concepts will be covered. As this project is in the domain of computer science. The music theory components will be explained as plainly as possible and their purpose in the application will be clearly defined. The most important component of this, that we will discuss in this paper is the key of a piece of music. We will briefly cover the key here as it is an integral part of understanding this project. The key of a piece is essentially a group of notes that make up a scale. The root note of the key is the same as the name of the key. So, in the key of A the root is also A. These group of notes complement each other and sound "pleasing" to the ear when played in succession.

The Core objectives of this project are as follows.

- To explore the possibilities of music generation through machine learning. Ie. what a machine can and cannot do (if anything) in terms of creating music.

- To generate a musical composition that a first-time listener wouldn't know it was written by a machine.

- To generate a Composition within a reasonable time.(the same if not less than the time it takes to play the composition.)

- To produce an output that is in an editable form so that potential users can manipulate it as they see fit.

This Projects finished output will be musical compositions. The quality of this music is a difficult thing to quantify or apply metrics to as weather a piece of music is "good" or not is usually highly subjective. One-persons favorite song could be mindless noise to someone else. For this reason, rather than using how "good the music is" as a metric we will use the following.

- Musical correctness (adherence to music theory rule and constraints).

- Time taken to generate the musical composition.

- Complexity of the generated musical composition (ie. Not impressive or interesting if just repeats the same three notes, even though they could technically be in key and follow all the rules.)

- Variance between compositions (ie. dose not generate repeatedly similar pieces.)

While the primary objective of this project is to create a neural network-based model that can help us generate musical compositions that are indiscernible from human compositions. I believe its greatest strength could be as a tool. Rather than take the output of this program as the finished product a musician could use it as an endless set of music "writing prompts". As anyone who as ever tried to write music, paint or even write knows, starting can be the hardest part. This project could be used as a tool to help alleviate this problem for musicians. The goal of having an editable output would also allow musicians who produce music using a computer to directly take the generate composition and chop and change parts of it for their own compositions.

This Document Will Contain the following Chapters.

Methodology.

- This section will contain the Methodology used in the Research, Design and implementation phases of the project

Technology Review.

- This Section will contain a review of each of the technologies to be used in this project and why they were used.

System Design

- This section will contain a detailed explanation of the overall system architecture or the "How" of the project covering all the components of the project and how they are used in conjunction with each other to complete our goals outlined above.

System Evaluation

- This Section will evaluate the completed project against the objectives and goals outlined above. It will also highlight any oportunites or limitations discovered during the implementation of the project.

Conclusion

- This section will summarize the context and objectives of the project and highlight any findings discussed in the System evaluation section.

Repository

https://github.com/Uamhan/mBot

The URL above leads to the GitHub repository for this project. We will now list the contents of this repository with a brief description of its contents.

The Classical Folder Contains all training data for the neural network.

The src folder contains all of the source code for the program.

The README contains a brief outline of the project and project setup instructions.

A copy of this dissertation.

A brief 2-minute video showing the end programe working.

Methodology

In this section we will discuss the methodology used in the research design and implementation of this project. We will first discuss the overall project plan, the research phase of the project, followed by the design phase. Finally, we will discuss the methodology used in the implementation of the project.

The overall plan when approaching this project was to firstly conduct thorough research on the domains that would be explored during the design and implementation of the project, to acquire a deeper understanding of the subject matter and how best to approach the implementation. Secondly to Determine the requirements for the design phase. Third to create a rough design and attempt to implement a prototype off of this design to quickly discover any pitfalls or misconceptions early in the development. Fourth to refine this design based on the knowledge acquired in the first iteration. Fifth and finally to implement this refined design or to yet discover more pitfalls and repeat the process of refining the design and implementation. This iterative approach would allow for discovery of misconceptions or unforeseen pitfalls early in the development process. An empirical approach was taken in the design and implementation phases of the project by experimenting with the neural network via adjusting the weights of the network, manipulating and adjusting the training and validation data, changing the model of the neural network or even changing how we use the output of the network to generate music. Scrutinous evaluation of the results from these experimentations was then used to decide on how to get the best results from our network.

The First phase of this project was the research phase where extensive research in the domains of neural networks, machine learning and Music Composition theory was undertaken. To Implement the use of Neural networks through machine learning in generating new musical compositions. First these building blocks needed to be understood. Through this research it was discovered that in order to achieve the goal of generating new compositions firstly a vast amount of training and validation data was needed. For the purpose of this project one genre was chosen. Specifically, Classical music as it has a vast array of openly available music that is in the public domain and is open to free use. The model could be trained on more genres all that would be required is to provide a new set of training data and use the same training methods. During this research it was discovered that the most logical approach to acquiring and using this data was not to use audio files at all but to instead use MIDI files (Musical Instrument Digital Interface). These files are digital representations of the music notes and how they are played rather than raw sound. They can be seen as a digital set of instructions on how to play a composition. Rather than the sound of a rendition of the composition. This provided many advantages. They are far smaller in file size than their audio counterparts. They are much easier to manipulate than raw sound waves as they follow a form of syntax that has hard definitions of what note is to be played, when and for how long. It can be thought of as a list of notes to be played in a given order. This

allowed for conversion of these note values into number representations making the preparation of the data for the neural network more streamlined than raw audio files. The chosen method of generating a melody from a neural network was also discovered through this research. The method decided upon entails calculating the probability of a given note proceeding a string of previously played notes. Acquiring a starting string of notes and generating new notes by selecting the most likely next note. Music Compositional Theory research was also undertaken to obtain information on how a classical piece was typically put together so as to try and implement similar methodology's when training and testing the neural network.

The second and Third Phase of this project, design and implementation where approached iteratively. An initial design was created and then implemented. The outcome of these steps where then evaluated and iteratively repeated.

The first iteration of the design phase. The goal of the first iteration of the design phase was to take the initial Conception of the project acquired from the research phase, develop a set of requirements and create a rough first draft design of the project. The initial requirements where to acquire a training a validation set of classical MIDI files. An initial neural network model. A method of preparing the MIDI files for use with the neural network. A means of generating a MIDI file from the output of the neural network and finally a means of playing this MIDI file back to the user. From these requirements the initial design was created. This initial design consisted of parsing the Training Data midi file notes into numerical representations. Training the neural network with these parsed notes. Using the neural network to generate a string of new numerical representations of notes parsing these notes back into a midi file and finally playing this new midi file. The next step was to implement this initial design.

The first iteration of the implementation phase consisted of writing the code to parse the MIDI file data into numerical representations. Creating the neural network model and training it on the parsed MIDI data. Writing the code that would generate a new string of numerical note representations based on the output of the neural network. Writing the code that would Parse this string of notes back into a MIDI file and finally play this new MIDI file. The first iteration was successful in generating a new MIDI file and playing it. This MIDI file consisted of the same note being played repeatedly for its entire duration. Through evaluation of the written program it was discovered that the problem lied in the training data for the neural network. A much larger set of training data was required than initially planned for. It was also discovered that the first implementation also required the neural network to be trained each time the program was run. This information was reflected upon and we returned to the design phase for a second iteration.

In the second iteration of the design phase the goal was to rectify the short comings of the first iteration and take what was learned to decide on any improvements that could be made to the design. Such as finding a means to save the trained model weights in a separate file so that once trained the model could repeatedly generate new

compositions without further training. In this second iteration of design it was also decided that the user should have more control over the melody they wished to generate. This meant implementing a means that the user could select the tempo, key and tonality of the composition and the output from the neural network would be manipulated to meet these specifications before playing back to the user.

In the second iteration of the implementation phase These new requirements were coded. The neural network was trained using a much larger training and validation data set. The output of the neural network now producing some semblance of a music composition.

These two phases where repeated until a final Design and implementation was complete. A vast amount of the time taken by this iterative process was in the creation of the most optimal neural network model possible. This was achieved via experimentation in parameters, manipulation of the training data and how the output of the network was used to create a composition. This iterative process led to the finalized design and implementation described in the System Design section of this dissertation.

For validation of the output of this program it was decided that the best means of validation would be that the composition adheres to Music Compositional theory constraints as whether the composition is good or not is highly subjective in the domain of music. This required that the Composition produced by the program followed Compositional constraints such as staying within its given musical key i.e.. Didn't play any off notes.

GitHub was the chosen means of version management for this project. Although the project solely consisted of one member. It was decided that GitHub would provide an effective means of keeping an accurate version history.

Problems in this project where approached in an iterative manner as discussed in the iterative approach to design and implementation above. Each iteration attempting to solve the problems of the previous one.

Technology review

In the Technology Review Section, we will discuss the technologies used in the implementation of this project. This section will be split it to sub sections each describing a separate Techology at a conceptual level, why it was chosen for this project and finally where and how it was used in the implementation.

Python Programming Language

The Python Programming language was chosen as the main Programming language for this project.

The python programing language is an interpreted high level, general purpose programming language. It was initially created by Guido van Rossum in 1991. Pythons main design philosophy puts heavy focus on code readability [1].

Python is a dynamically typed and garbage collected language meaning that it executes many common programming behaviors at runtime as opposed to at the time of compilation [2]. This allows for generation of new code, objects and definitions during the running of the program making it a strong choice for machine learning based projects.

Python is known for allowing the creation of readable and maintainable code.
Python allows for this in multiple ways firstly through its less is more structure. An example of this would be the lack of an end line operator. Usually at the end of a line of code it is required you define the end of the line with the ";" operator. In python this is not needed, it simply takes that a new line denotes the end of line. Another example of this less is more structure is that in python indentation is used instead of brackets for all structures. This allows for cleaner more human readable code. Secondly python allows the use of English keywords in the syntax for common programming concepts such as using the key word "or" instead of the usual operator ||. All of these examples and more can be found in the python documentation [3].

Another Strength of the python programming language is its incredibly robust standard library. This large library allows you to select from a vast array of modules that suit your needs, without the need to install new library's. These modules cover a wide variety of commonly needed functionality. From Text processing, numeric and mathematical functionality, functional programing functionality file and directory access, data persistence and far more. For a full list of modules see [4].

To Further accentuate the vast standard library provided by the python programming language, Python is an open source programming language and as such has access to a

wide variety of open source frameworks and tools. Some of which will be discussed later in this technologies review. The access to these library's provides functionality that not only greatly reduces development time but also costs nothing to the developer making it a great choice for a student project. Python provides incredibly simple access to this vast array of open source library's through the pip package management system [5]. Usually requiring no more than a one-line command to fully intergrade the library into your python environment.

As the main programming language of this project it was used exclusively for the development and implementation of this project. To reiterate it was chosen for its human readable syntax its extensive standard library and finally its vast array of open source libraries.

Python was also chosen as the primary programming language as a means to strengthen python programming skills as it is one of the fastest growing programming languages at the moment as discussed in K. R. Srinath's paper on this topic [6].

TensorFlow

TensorFlow is an end-to-end open source platform for machine learning it provides a comprehensive, yet flexible set of tools and libraries, that allow developers to build and deploy powerful Machine Learning powered applications.[7]

The implementation of machine learning is a complex process and tool kits such as tensor flow provide the developer with a vast array of resources to streamline the machine learning aspects of their projects. Tensor flow provides a simple means of model building with multiple levels of abstraction so that the developer can decide what level of control they need over the machine learning model. It also allows the developer to create one model and distribute it on multiple different hardware configurations without out having to change the machine learning model. The specific TensorFlow api used for this project was the Keras API which will be further discussed later in this technology review section.

The Keras toolkit provides the means for rapidly developing machine learning models. This was a much-desired attribute for this project as creating an effective machine learning model for the generation of musical compositions required a vast amount of experimentation. The ability to rapidly prototype models for this experimentation allowed for the best model to be created in the given timeframe.

TensorFlow is an opensource project and is readily available for the python programming language providing straightforward integration into the python development environment.

The TensorFlow Toolkit also provides an extensive amount of documentation on all functionality it provides [8] along with comprehensive learning materials on all functionality provided by the toolkit. This level of documentation was of great help when experimenting with the integration of music and machine learning.

Tensor flow is a scalable toolkit that provides the means to take on large scale machine learning projects yet remains highly effective in smaller projects such as this one. This would allow for expansion of the scope of the project in the future.

To reiterate TensorFlow is a highly scalable machine learning toolkit with a variety of API's to work with when developing a machine learning model, it was chosen for this highly scalable nature. Its facility's for rapid model prototyping and extensive documentation.

Keras

Keras is a high-level neural network API Written in python and capable of running on top of the TensorFlow Framework. It is focused on allowing fast experimentation. With an emphasis on going from idea to result as quickly as possible. Keras allows for quick prototyping and supports recurrent neural networks running on either CPU or GPU hardware.[9]

The Keras API is designed for user friendliness making it as humanly readable as possible which provides strong cohesion with the principles of the python programming language. It has a strong focus on modularity allowing the developer to select the sections they need for their project. Also allow for quick integration of new modules as they are needed with very little refactoring of existing code. As it is written in python it provides simple integration into a python programming language environment making it a strong choice for this project.

Keras provides straightforward means of CPU and GPU integration in the training process. This GPU integration was key factor in the selection process for this project as the machine learning model used for the generation of music in this project required a vast amount of training data. Initial prototypes where tested with CPU based training and required Training times of 40+ hours. This was an unacceptable amount of time given the experimental nature of the project. Once GPU integration was implemented training time was reduced to 3-4 hours. Making prototyping models and experimentation more efficient by a factor of 10. For this reason alone, keras could be seen as a strong choice for the Machine learning API of choice for this project.

Keras provides extensive documentation on all of the API's functionality providing an excellent resource for the experimentation required in developing a machine learning model that could generate a musical composition.[9]

Keras provides a means to define a machine learning model line by line by defining a model and adding layers of nodes one by one with keras methods taking arguments to define how each layer behaves in the model.

Keras also provides extensive functionality to visually represent the metrics of a machine learning model.

To reiterate Keras is a high-level neural network development api with a focus on user friendliness developed in the python programming language backed by large corporations such as Amazon, Microsoft, Google and Nvidia. It provides a rapid means of machine learning model prototyping making it a strong choice for the experimental nature of this project in generating musical compositions through machine learning.

Anaconda

Anaconda is an open source python distribution for data science related programming that aims to simplify package management and deployment. It provides a strong starting point for any python environment related to data science development. It contains more than 1400 opens source python libraries ready to install as needed. [10]

The anaconda package provides a straightforward means of creating a python runtime environment for data science related tasks and as such was a strong starting point for the implementation of this project it provides a simple means of installing TensorFlow and keras as discussed above. It also contains various library's for manipulating data to allow integration with machine learning.

The anaconda package was chosen as a starting point when creating the development environment for this project as it contained many of the necessary library's in data preparation for the machine learning models and also provided a straight forward means of setting up the TensorFlow and keras modules discussed above witch where an integral part of this project.

Anaconda provided an easily replicable development environment for the purposes of this project it allows the compilation of the source code for this project with only minor modifications for this project.

Music21

Music21 is an opensource python library that acts as a toolkit providing an extensive amount of functionality the creation, manipulation and reading of MIDI data.

MIDI (Musical Instrument Digital Interface) files being the means of music notation chosen for the implementation of this project. These files are digital representations of the music notes and how they are played rather than raw sound. They are a digital set of instructions on how to play a composition.

The Music21 library Contains a comprehensive selection of classes representing common music structures, such as the note object that represents a musical note. Notes are represented by the letters A through G. they can be either neutral, sharp (denoting that the note is a semitone up from the neutral or half way between the note and the note above it) and flat being the same as sharp but between itself and the note below it instead of above. The chord Object represents a music chord being constructed of note objects much like a chord is constructed of notes. The strength of Music21 isn't the music notation itself as this would be easily replicable. The strength lies in the functionality associated with these classes.

Such functionality includes the ability to manipulate the pitch, duration and accents of these notes. Manipulating the duration of a note being a fundamental concept for this project. To implement note data in our machine learning we must have a node for each possible input. With duration tied to each note in a midi file the number of possible inputs is undefinable as a note could be of any duration. Music21 provides us with a means of manipulating this duration information and as such artificially generating it. This means we can strip the duration from the training data. Leaving us with a string of notes with no context other than the order they are played in. With duration removed from the notes we can now define the possible inputs for our neural network model. The human hearing range is between 20hz and 20khz.[11] the lowest note in this range C0 and the highest being C10 this infers that we have 10 keys that can be heard in music with 12 notes per key gives us 120 possible notes. A much more manageable set of possible inputs. The problem with this lies in the fact that our output will now have no duration and simply be a string of notes with no context. This is where Music21's duration functionality comes in it allows use to add the duration back onto our notes dynamically using any means we see fit for the rhythm of or generated music composition.

Music21 allows us to manipulate the tempo of a melody. The tempo being the speed of which the composition is played. This can be used to adjust the speed of the composition after generation to fit the user specifications.

Music21 is an integral part of this project as it provides the means to read the MIDI training data used to train the machine learning model. This data is read in using music 21 and parsed into a dictionary of numerical representations of the music21 note objects these are then used to train the machine learning model. Once trained the output of the model is then converted back into music21 note objects referencing the dictionary. This string of music21 notes can then be converted back into a MIDI file that can then be played.

In short Music21 is the chosen library to implement the musical notation manipulation and playback for the project. Providing a means to integrate the use of MIDI files with our machine learning model.

NumPy

NumPy is the fundamental package for scientific computing in python. It contains among other things a powerful n-Demensional array object essential for machine learning related programing, useful linear algebra, Fourier transform and random number capabilities all of which are extensively used in machine learning.

The n-dimensional array objects are a core component of any machine learning as they are used to represent the data input and feed through the machine learning model. The transform functionalities are also extensively used in manipulation and normalization of data for the machine learning model.

The random number capabilities of NumPy are also of great use in a variety of programs most notably for this project the ability to randomly select a piece of data for manipulation.

NumPy has extensive documentation on all the functionality it provides providing the developer an excellent reference when using the NumPy library [12].

In short, the NumPy library provides much needed scientific computing functionality in terms of the n-dimensional arrays used in machine learning.

Pygame

Pygame is a cross platform set of python modules design for creating video games. It includes libraries designed for computer graphics and sound libraries for the python programing language.

The pygame library contains extensive sound functionality most notably for this project it contains MIDI file playback functionality. This is implemented in this project in the playback of the generated music composition.

Pygame provides a lightweight solution to MIDI file playback to the user for this project and allows for high audio quality to best represent the compositions generated by our machine learning model.

Pygame provides extensive documentation allowing for straightforward implementation of MIDI file playback functionality.[13]

Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the DeFacto standard GUI toolkit.

Tkinter provides a diverse set of tools of GUI creation and manipulation in Python and provides all the necessary tools to create a GUI for a python program. If being the DeFacto GUI standard for python wasn't enough of a reason for choosing it as the GUI toolkit for this project it provides a light weigh means of creating a GUI with extensive functionality, covering all of the GUI needs for this project.

This project primary focus is to through the use of machine learning, generate musical compositions provided training data. The user interface section is simply a means of having some control of the output of the program and playing the audio back to the user.

The GUI aspect provided by Tkinter allows for the user to select input parameters to manipulate the output of the machine learning model by selecting the key and tempo of the composition. These parameters are then used to manipulate the output of the machine learning model.

Tkinter was chosen because it provides a straight forward means of creating an interactive GUI that has direct integration with the backend of the program.

Review

All of the Technolgies discussed above are used in conjunction with each other as the building blocks of this project and provide the functionality needed to implement a machine learning model that can generate new musical compositions from a set of training data. Everything from preparing the input data for training to the creation of the machine learning model to the generation of the MIDI file and the playback of this file to user.

System Design

In this section we will discuss the overall system design of the project, it will be comprised of two sub sections, the system architecture overview section will give a synopsis of the overall program from start to finish and the Component section which will give a detailed explanation of each component of the system and what roll in the project it fulfils.

System Architecture Overview.

As Discussed in the introduction the goal of this project is to through the use of a long short-term recurrent neural network generate a musical composition. A large collection of training data is needed for the training of the neural network and as such is the first step. For the current implementation the chosen genre of music is Classical. Classical music was chosen as the genre has a vast source of open domain music to use. The file type that was chosen to represent this music was MIDI (Musical Instrument Digital Interface). MIDI files are digital representations of music notes and how they are played as opposed to raw audio. They can be seen as a digital set of instructions on how to play a composition, rather than an audio rendition of the composition. MIDI files were chosen as they are already in a format suitable to the manipulation required for our machine learning model. The MIDI files where acquired through various MIDI distribution websites. The next step was to create a machine learning model that could learn from this collection of data. The model used in this project went through many iterations before producing acceptable results. The final model is described in detail bellow in the component section. With model and a set of data the next step is to prepare the data for the machine learning model. This is accomplished by first converting the MIDI data into a format useable by our machine learning model. This was accomplished by creating a dictionary of note values with a corresponding numerical value for each note. Now that we have the notes in a format is usable by the machine learning model, we parse all of the training data through this dictionary into a large string of our new numerical representations of notes. From our dictionary we can get the total number of different notes in our training data number is used as our number of input nodes in our model. This data is then fed through our machine learning model. The training consists of over 100 iterations through the entire dataset adjusting the weights to represent the likelihood of a note given a string of previous notes. Once trained we have a model that we can use to predict the likelihood of a note given a previous string of notes. This is an integral concept in the generation of new notes. To generate new notes, we randomly select a starting point in the input data. We take the next 100 notes from

this section. These notes are used with the model to predict the most likely next note. This new note is appended to both our output list and the 100 notes we used to generate the note. The first note in the list of notes we used to generate the new note is then removed once again leaving us with a list of 100 notes with our generated note at the end. This process is repeated for each new note we want in our composition. Currently set to 250 notes to generate this roughly equates to 2 minutes of audio at a standard tempo. During the generation of the note's duration values are assigned to them. We now have a string of numerical note representations that represent our composition. This string is then parsed back to MIDI representations of the notes using the dictionary we created at earlier. This composition is then transposed to the desired key selected in the GUI. This manipulates the midi note objects by changing their values to the equivalent values in the desired key the tempo selected in the GUI is then assigned to the MIDI string and finally This string of midi notes is written to a MIDI file. Through the GUI the user can now playback or save this MIDI file to their machine. The GUI consists of 4 combo box elements with appropriate values for the user to select regarding the temp of the song the key of the song the tonality of the key and the keys accidental (being neutral, sharp or flat). The Gui also contains three buttons. Generate which executes the generation code described above. Save which opens the machines file directory where the user selects where to save the MIDI file and under what name and lastly a play button that will play the Generated MIDI file to the user.

Components

Model

The model is a core component in any machine learning project. The model is the network itself. It is the mathematical representation the series of nodes that make up our neural network. In this section I will discuss how our model create and what it comprises of.

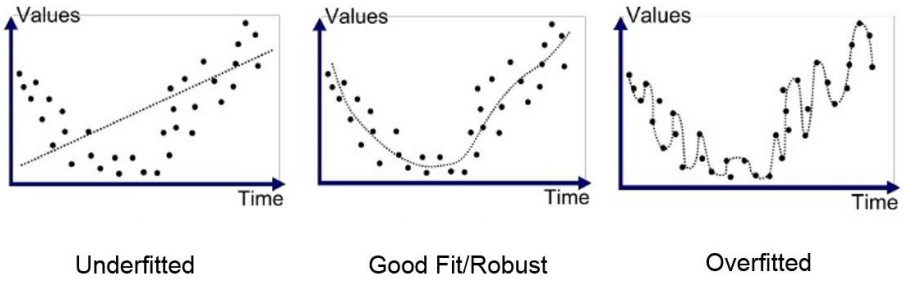
For the creation of this model the Keras API was used as described in the technology review section. The model is initially defined as a sequential model this infers that each layer of nodes executes one after another. We then add layer by layer until we have a fully defined model.

This model is a long short-term Memory or LSTM model it is a recurrent neural network that is comprised of an array of internal gates. Unlike other recurrent neural networks, a LSTM network’s internal gates allow the model to be trained using back-propagation this allows the model to learn based on the results of previous layers. Each layer is comprised of a serious of internal cells or nodes.

The layers of the model used in this project are as follows.

Input LSTM layer as the first layer of a sequential model we need to specify the shape of the input. This input shape is determined by the input data and is covered in the input data processing section bellow. We also need to define the number of starting nodes for this layer. For the purposes of this model we go with a standard 256 nodes. We also set the return sequence to true. This allows our model to access the hidden state output for each timestep.

Dropout layer (0.3) The next layer is a dropout layer with a parameter value of 0.3 this tells our Keras model to ignore 30% of the nodes when determining the next layers weights. This 30% is randomly selected at each iteration of training and prevents overfitting. Overfitting refers to when the model is trained to strictly to the given input that it provides very accurate results on the data it was trained with but will be very inaccurate when dealing with any new values. The diagram bellow illustrates this point further.



LSTM layer 512 return sequence true. adds another neural network layer with 512 nodes also with the return sequence set to true. This acts much like our first input layer but without the input shape definition as keras will infer the shape between layers.

Dropout layer (0.3) adds yet another dropout layer as described above.

Dense layer (256) this adds a dense layer of 256 nodes much like the lstm layer but this time being dense implies that each node in this new layer is connected to every node in the previous layer.

Dropout (0.3) adds yet another dropout layer as described above.

Dense layer (possible notes) adds a final Dense layer as described above but this time with the number of possible notes as the number of nodes each node representing a note. The strength or weight of this node nenotes the probability of it being the most likely next note.

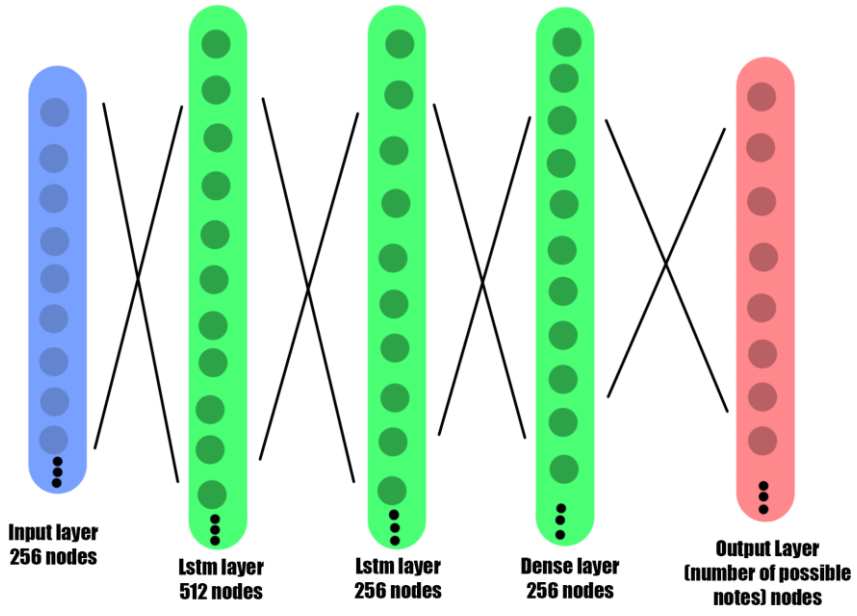
Add and activation of SoftMax this applies the soft max activation algorithm to our output nodes.

Finally, we compile the model with categorical cross entropy as the loss function and using the rmsprop optimizer.

The categorical cross entropy loss function is a SoftMax activation plus a cross entropy loss we use this as the loss function as it will train the neural network to output a probability over the number of possible outputs for each possible output. This gives us the probability of the next note that we need for the generation of new notes as discussed below.

The rmsprop optimizer is a version of the gradient descent algorithm used to efficiently calculate how best to adjust the model's weights after each iteration through back propagation.

A full visualization of the model can be seen bellow.



Input data

The input data or training data. For the machine learning model used in this project consists of 120 classical compositions ranging from short compositions to full symphony's written by predominantly by Johann Sebastian Bach, Ludwig van Beethoven and Frederic Chopin. The Compositions are all stored in MIDI files and are freely available to download as they lie within the public domain. This collection provided ample training data for the training of our machine learning model. Although 120 compositions may not seem like a lot when broken down into the individual notes for our model it roughly equated to 118,000 notes. More than enough for purposes of our project. This training data can be found in the directory named Classical within our repository (link available in the introduction). Classical Music was chosen as the genre for this project as it provided an ample supply of public domain training data, but the training procedure implemented in this program is designed so as to work with any genre provided enough data.

Input data preparation

To effectively use our machine learning model as described above. We need to properly prepare our data for use as input. Our raw input before preparation consists of a collection of Classical MIDI files. To prepare this data we iterate through each MIDI file parsing them into a midi stream using the inbuilt converter found in The Music21 library. We then separate this midi stream by instrument parts to get each individual note progression in that midi file each of this note are then appended to a list notes as a string value of the note. Once this has been completed for each MIDI file, we are left with one long list of string representations of the notes. We then create a sorted set of notes from this to get a list of all the pitch names from our training data. We then get the length of our note set to get the number of different notes. We then create a note to integar dictionary that maps each note to a number. This is done so that we can use numerical representations of the notes to train our model. We will also use this same dictionary to convert the output of the model back into note objects. We then fill a network input array with all of the numerical representations of the notes. We then reshape this array of notes into a format that the Keras model will accept this process is referred to as normalization. Once complete our data preparation is complete and we are ready to train the neural network.

Model training

This section will cover the model training. As defined in the sections above once we have the model created and the input data prepared The keras API provides us with the model.fit() method. This method runs the model training based on a series of arguments, for this project that training call is as follows.

```
model.fit(normalNetworkInput, networkOutput, epochs=100, batch_size=64, callbacks=callbacks_list)
```

This takes our normalized network input (our processes input data described above) and our validation network output. It runs for 100 iterations. With a batch size of 64 and stores any callbacks in our callback list.

This is a very resource intensive process and in the development of this project it was executed on a Nvidia 1070 8gb virtual random-access memory GPU and took roughly 4 minutes per epoch it took just over 6 and a half hours to train. This timeframe is in relation to the final model. Many earlier iterations required time to train and test some taking long that the six hours some less.

Note Prediction

This section will discuss how the trained model generates notes for our composition.

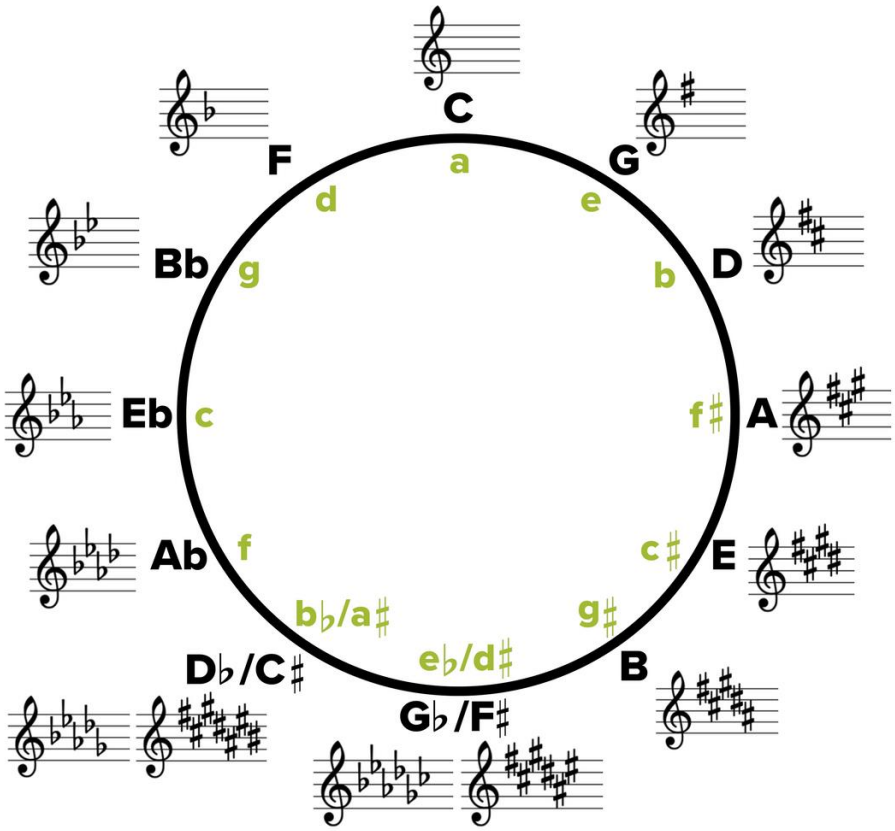
The first step is to acquire our input for the prediction to do this we randomly select a starting point in our list of input notes used to train the model and take the proceeding 100 notes as our data. We normalize this selection of notes to be used with the model much like when prepping the input data as described above. We then call the model predict method on this data. This returns us with an array of probability values for the likelihood of each note after this string of 100 notes. We then call the argmax method on this array of values to return us the most likely note to be played next. Two things need to be done to this value. First it is appended to the end of our string of notes used for prediction and the first is removed so we can calculate the next note in the next iteration and secondly. This value is converted back to a string representation of a note using the dictionary we defined and discussed above. and appended to a new list of generated notes. This process is repeated for each not we require for the purposes of this project the value of 250 was chosen as at a standard tempo it equates to roughly 2 minutes of audio. Once complete we have a list of generated note values represented by strings. These note values are then used to create a set of new Music21 note objects that are assigned an offset which represents the time between this note and the start. This value is used to assign the duration of each not. Once all notes have been converted to Music21 note objects this list of objects are feed into a MIDI stream object to be manipulated by the users input parameters as discussed below.

Output Manipulation

This section will discuss the manipulation of the composition once it has been generated as discussed in the previous section.

Once we have our generated composition. It needs to be manipulated according to the parameters the user selects from the GUI. These parameters are the key of the piece. Weather it is a major or minor composition the key accidental and finally the tempo of the composition. The Key signature of a piece is an amalgamation of the above excluding the tempo element. The key signature comprises of firstly of the letter denotation. With the possible keys ranging from A-G. This value is taken in directly for this example we will go with the key of A. the next input from the user is weather the key is minor or major in music notation and the Music21 library a major key is denoted with a capital letter and a minor with a lower case letter. For the manipulation of the key letter we simply apply pythons upper or lower methods to our key letter value depending on the user's selection. “A” representing the key of A major and “a” representing A minor. The next step is to denote weather the key is sharp neutral or flat. If the key is neutral, no modification is needed, and we are left with our key signature. A sharp value in music notation is represented by the “#” symbol and as such needs to be appended to the end of our key value so A sharp major would be denoted as “A#” and A sharp minor would be denoted as “a#”. If the flat option is selected, we need to append the flat operator to the end of the key value. In music theory a flat is represented by a lower case “b” so for the case of A flat major we would have “Ab”. A sharp note is having a semi tone up from the letter before it so A sharp is actually half way between A and B. a flat note is the opposite of this and is a semi tone down from the note, so A flat is half way between A and G.

Once we have our desired key signature. We can transpose the composition. In music theory transposition is the act of changing a group of notes from one key to the equivalent notes in another key. This can be achieved by changing the keynote (known as the root note in this context) to the note of the desired key. We take note of how many semitones (half notes) it took to get to this new note from our starting note and then apply the same transformation to every note in the composition. A common means of learning the method of transposition in music theory is a diagram known as the circle of 5ths as seen in the diagram bellow. It is comprised of an inner and outer sequence of notes going around a circle each note being a fifth of the previous note meaning 5 half notes between each you see how many jumps the first note of the composition requires to get to the desired key the for each subsequent note you find it on the circle and apply the same amount of jumps.



To achieve this in this project we take advantage of the Music21 library inbuilt functionality as described in the technology review section. We the Music21 implementation of the Krumhansl algorithm to determine the key of the generated piece. Get the distance or interval between the detected key and the desired key and apply the transposition method using this interval to effectively transpose the generated composition to its new key.

The final manipulation required is to change the tempo of the composition to the user specified value. This is achieved by taking the users desired tempo from the GUI as described below and assigning it to our MIDI stream before it is written to the MIDI file its self.

GUI

The Gui for this project was developed using the Tkinter framework as discussed in the technology review section above.

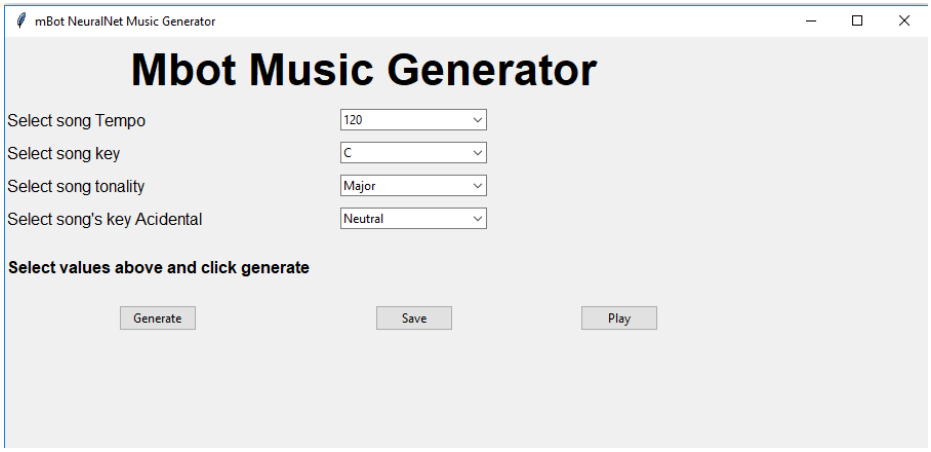
The GUI for this application provides a means for the user to select the Tempo, Key, key tonality and key accidental they wish the composition to be. It Provides a visual interface to select these variables from a series of combo boxes. The Gui also provides three buttons the first to generate the Composition, the second to save the generated MIDI file to their machine and third to play the audio file.

The GUI consists of four Combo Boxes the First Being Tempo. This tempo value is taken during the generation stage to assign a tempo to the generated composition.

The following three combo boxes work in conjunction with each other to form the final key signature that will be applied to the composition. The second Combo box element allows for the selection of the key from A-G. The Third Combo allows the user to select weather the composition will be in a minor or major key and the fourth combo box allows the user to select the accidental of the key. These three values are combined as described above in the output manipulation section to form the final key signature that is applied to the composition.

The GUI also contains three buttons. The first Button is to generate a new composition based on the combo box parameters described above. The second is to save the Generated MIDI file to the user's machine. Using Tkinters file dialog method to select a path to save to and then use Music21 MIDI write functionality to write the midi file to this selected location. The third and final button is used to play back the MIDI file to the user. It accomplishes this by calling the play method which in turn initializes a pygame mixer object with cd audio quality parameters. This mixer is then used to load the generated MIDI file and play it back to the user.

A bare bones prototype of the GUI can be seen bellow


mBot NeuralNet Music Generator
Mbot Music Generator
Select song Tempo
120
Select song key
C
Select song tonality
Major
Select song's key Acidental
Neutral
Select values above and click generate
Generate
Save
Play

System evaluation

In this section we will evaluate the completed project against the goals and metrics outlined in the introduction section of this document. First, we will go over the Goals and objectives one by one and compare in initial goal/objective to the completed project.

Goals/Objectives of the project

The first Objective outlined in the Introduction was the following.

- To explore the possibilities of music generation through machine learning. Ie. what a machine can and cannot do (if anything) in terms of creating music.

From the undertaking of this project it was assumed that since music is created to adhere to the confines of music theory. Weather the musician is aware they are using it or not. That it would be replicable by a computer as it is simply a set of rules. Through Research and the implementation of this project this assumption was found to be true with a few minor caveats. It is true that since the creation of a musical composition follows a set of rules, some genres use these rules as a set of guidelines rather than hard set rules. Some masterful implementations of this can even purposefully break rules (i.e.. playing off notes, or even notes out of rhythm) to add accent to a particular feature of the music. This while not impossible is a much more difficult thing to achieve with a machine as there is no hard-set rule to when this is appropriate if it all. It was also found through the implementation and research for this project that rhythm isn't as straight forward as the notes themselves. This is due to that fact that while there are rules to how the rhythm of a composition should be constructed. The main defining rule of rhythm is that all of the components of a piece should be in rhythm with each other it is at the musician's discretion if they wish to change this rhythm at any point in the piece some artists choosing to do so regularly. While not a common feature in musical compositions it is a difficult feature to replicate.

The Second Objective outlined in the Introduction was the following.

- To generate a musical composition that a first-time listener wouldn't know it was written by a machine.

This was one of the core measurable goals that would define weather the project was a success or not. For the most part it was a success. Some compositions where less successful than other. But for the most part the vast majority of the compositions

performed excitingly well. An array of compositions where generated and fed to various listeners without context and very few noted that there was anything unusual about the compositions.

The Third Objective outlined in the Introduction was the following.

- To generate a Composition within a reasonable time. (the same if not less than the time it takes to play the composition.)

The goal of this objective was to produce the compositions in a timely manner so as to be able to iterate through them quickly to find one you like or would have use for. This objective was a success as it takes between 30-45 seconds to generate 2 minutes of audio at a standard tempo of 80 beat per minute.

The Fourth and final Objective outline in the Introduction was the following.

- To produce an output that is in an editable form so that potential users can manipulate it as they see fit.

This goal was achieved to great success through the use of the MIDI file format extensively discussed in this document. This file format allows the user to open up the generated MIDI file in any midi editing or manipulation software to manipulate as they fit.

Metrics

This section will reiterate the metrics of success discussed in the introduction and evaluate the end product of the project against them.

The First metric outlined in the introduction was the following.

- Musical correctness (adherence to music theory rule and constraints).

The end product of this project performed excellently in this regard as the neural network was trained using musical compositions that followed these constraints to resulting output also followed these constraints with great accuracy.

The Second metric outlined in the introduction was the following.

- Time taken to generate the musical composition.

The end product of this project performed sufficiently in this regard as discussed above the final implementation produced 2 minutes of audio at a standard tempo of 80

beats per minute in between 30-45 seconds. This is s sufficient result as it generates the audio at less than half the time it takes to play it back. This could be improved by further refinement of the generation process but would likely require far more powerful hardware than was available for this implementation.

The Third metric outlined in the introduction was the following.

- Complexity of the generated musical composition (ie. Not impressive or interesting if it just repeats the same three notes, even though they could technically be in key and follow all the rules.)

The end product of this project performed sufficiently in this regard as it produces musical compositions with little repetition from start to finish. With the rare outlier that doesn't vary too much. From the roughly 200+ compositions generated for the purposes of testing the output of this project only 2 of which contained long strings of repeated notes. This equates to an occurrence rate of less than 1% witch for the purposes of this project is a success.

The Fourth metric outlined in the introduction was the following.

- Variance between compositions (ie. dose not generate repeatedly similar pieces.)

The end product of this project performs very well in this regard as discussed in the system architecture section above the notes are generated from a starting point within the training data as reference. With over 118000+ notes in the training data the likely hood of similar pieces being generated is acceptably low. While still a possibility, it is also a possibility in human created music with a relatively small set of possible notes one can play.

Overall the technologies used for the implementation of this project (discussed above in the technology review section.) performed as expected if not better than expected in some cases. The main limitation of this project was pure hardware processing capabilities. The machine used to develop this project contains a Nvidia 1070 8gb virtual Ram GPU. This was used for the training of the neural network model. While a very powerful GPU for general use the training of a neural network is incredibly demanding, and a training session of our model took approximately 6 and a half hours. This meant that each iteration of the model took at the very least 7 hours to test due to training time. With the use of more powerful hardware this training time could have been greatly reduced allowing for more iterations and experimentation with the neural network model. It would have also allowed for far larger training data sets which could have produced more interesting results. This said the end product of this project satisfied the goals outlined in the project to a satisfactory level.

Conclusion

The objectives of this project where to generate a musically correct piece of music that had a sufficient complexity in a timely manner that didn't produce regularly repeating results. This as a whole was a success. The final implementation produces 2 minutes of audio at a standard tempo of 80 beats per minute in 30-45 seconds with no repeat results and a less than 1% occurrence of insufficient complexity in the composition. As a whole I would consider this a great success.

Through the research and implementation of this project a great deal was learned in the domain of neural networks and how they can be used in conjunction with music. Some points of note where that melody generation was achievable as shown by this project but the complexity of generating complex rhythms for the music generated while achievable would require vastly more hardware power than was available during the implementation of this project. For the sake of training time the duration of notes during training needed to remove and added back onto the generated note. Ideally with enough processing power we would be able to leave this duration part of the note on for the training stage. Providing the neural network, a means to generate the rhythm of the composition. But even if we were to simplify the duration to even the most basic of lengths between the notes it would increase the number of training inputs by a factor of 16. This would require either a far greater amount of time to implement or far more powerful hardware neither of which were available during the implementation of this project.

Outcomes

- A neural network model trained with classical music that can generate new notes.
- A program that constructs these generated notes into a complete music composition.
- User friendly GUI to interact with the program and produce customized results.
- Further understanding of neural networks, music theory and the python programming language along with all libraries used in the implementation of this project.
- Further insight ascertained in the area of generation of "creative work" I.e. music, art, literature. Through computing.

Opportunities for further investigation include the exploration of the possibilities of what can be achieved in terms of rhythm generation given more processing power or time as discussed above. Further opportunities include the exploration of generation of

musical compositions in other genres of music. As this project focuses on classical music as outlined previously in the document do to the open domain nature of classical MIDI files. I believe this area of research contains a vast array of possibilities and will become ever increasingly accessible for experimentation as the access to processing power increases.

References

[1] Python Programing language Wiki, Available:
https://en.wikipedia.org/wiki/Python_(programming_language)

[2] Dynamic Programing, Available:
https://en.wikipedia.org/wiki/Dynamic_programming_language

[3] Python Documentation, Available:
https://docs.python.org/3/

[4] Python standard library, Available:
https://docs.python.org/3/library/

[5] Pip package management, Available:
https://pypi.org/project/pip/

[6] K. R. Srinath, Python – The Fastest Growing Programing lanaguage IREJET, vol. 4, issue 12, page , Dec-2017.

[7] Tensorflow Description , Available:
https://www.tensorflow.org/about/

[8] TensorFlow Documentation , Available:
https://www.tensorflow.org/api_docs

[9] Keras info, Available:
https://keras.io/

[10] Anoconda Description, Available:
https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

[11] human hearing range, Available:
https://hypertextbook.com/facts/2003/ChrisDAmbrose.shtml

[12] NumPy Documentation, Available:
https://docs.scipy.org/doc/

[13] Pygame MIDI documenetation, Available:
https://www.pygame.org/docs/ref/midi.html