**A Major Project Report (CS801PC)**

on

**MUSIC GENERATION USING DEEP LEARNING**

*Submitted*

*in fulfillment of the requirements for the award of the degree of*

### Bachelor of Technology

in

### Computer Science and Engineering

by

### Mr. Chinta Aditya Shashank

**(16261A0504)**

Under the Guidance of

### Dr. C.R.K. Reddy

**Head of the Department**

**Department of Computer Science and Engineering MAHATMA GANDHI INSTITUTE OF TECHNOLOGY**

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

Kokapet(V), Gandipet(M), Hyderabad.

Telangana-500 075.

**2019 - 2020**

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University Hyderabad) GANDIPET, HYDERABAD – 500 075. Telangana

# 

# CERTIFICATE



This is to certify that the project entitled **“Music Generation Using Deep-Learning”** is being submitted by **Chinta Aditya Shashank** bearing **Roll No: 16261A0504 in** partial fulfillment for the award of B Tech in Computer Science and Engineering to **Jawaharlal Nehru Technological University**, Hyderabad is a record of bonafide work carried out by her under our guidance and supervision.

The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

|  |  |  |  |
| --- | --- | --- | --- |
| Supervisor |  | Head of the Department | Principal |
| **Dr. C.R.K. Reddy** |  | **Dr. C.R.K. Reddy** | **Dr. K. Jaya Sankar** |

Professor Professor Professor

### External Examiner

# DECLARATION

This is to certify that the work reported in this project titled “**Music Generation Using Deep-Learning”** is a record of work done by me in the Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Hyderabad.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the work done entirely by me and not copied from any other source.

## CHINTA ADITYA SHASHANK

## 16261A0504

**ACKNOWLEDGEMENT**

I would like to express my sincere thanks to **Dr. K Jaya Sankar, Principal MGIT**, for providing the working facilities in college.

I wish to express my sincere thanks and gratitude to **Dr. C R K Reddy, Professor and HOD**, Department of CSE, MGIT, for all the timely support and valuable suggestions during the period of project.

I am extremely thankful to **Dr. C R K Reddy, Professor & HOD, Dr. M. Sreevani, Associate Professor**, Department of CSE, MGIT, major project coordinators for their encouragement and support throughout the project.

Finally, I would also like to thank all the faculty and staff of CSE Department who helped me directly or indirectly, for completing this project.

## CHINTA ADITYA SHASHANK

## 16261A0504

|  |  |
| --- | --- |
| **TABLE OF CONTENTS** |  |
| Certificate | i |
| Declaration | ii |
| Acknowledgement | iii |
| List of Figures | vi |
| List of Tables | vi |
| Abstract | vii |
| **1. Introduction** | 1 |
| 1.1 Motivation | 2 |
| 1.2 Problem Statement | 2 |
| 1.3 Proposed System | 2 |
| 1.4 Requirements Specification. | 3 |
| 1.4.1 Hardware Requirements | 3 |
| 1.4.2 Software Requirements | 3 |
| **2. Literature Survey** | 4 |
| **3. Music Generation stratagem** | 7 |
| 3.1 System Architecture  3.2 Tools  3.3 Machine Learning Modules  3.3.1 Music21 Library  3.3.2 Pickle Library  3.3.3 Keras Library  3.4 Dataset | 7    8  9  9  10    11    11 |
| 3.2 Dataset Preprocessing | 12 |
| 3.3 Algorithm      4. [Training and Results 1](#_TOC_250008)9   * 1. [Training 1](#_TOC_250007)9   2. [Results 22](#_TOC_250005)   5. [Conclusion and Future Scope 25](#_TOC_250004)  [Bibliography 26](#_TOC_250001)  [Appendix-Source Code 27](#_TOC_250000) | 3.3.2 13      17    17    19    21  22      23 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Figure 1.1 | Music Generation in the past | 1 |
| Figure 1.2 | Music Generation in the contemporary | 1 |
| Figure 3.1 | Architecture of The Neural Network Model Designed | 7 |
| Figure 3.2 | Anaconda Navigator | 8 |
| Figure 3.3 | Jupyter Notebook | 9 |
| Figure 3.4 | Chord Sequences Generated using Music 21 Library | 10 |
| Figure 3.6 | Notes Extraction from the input data | 13 |
| Figure 4.1 | Training a model | 17 |
| Figure 4.2 | Creating a Model | 18 |
| Figure 4.3 | Generation of a piano Midi file | 18 |
| Figure 4.4  Figure 4.5 | Conversion of the predicted output into midi format.  Output File Created | 19  20 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Table 2.1 | Literature Survey | 6 |

**ABSTRACT**

Use of deep learning techniques for generating music has gained popularity in the recent years owing to the high amount of compute power being available and the evolution of deep learning architectures which are well suited for learning patterns from sequential data.

In this project, our goal is to generate musical notes that will follow a given input primer sequence such that the entire generated musical sequence sounds continuous and melodious. For this, we leverage Music21, Musicgen, Play, Tqdm and propose new methods for data ingestion on Long Short-Term Memory (LSTM) based models. The proposed system is used for Deep-learning approach which feeds in extra information about the targets into the model to aid the prediction process, the observation that the melody is relatively independent of the starting note, and the difference in pitches of notes from the start note preserves the characteristics of a song in a compact manner, amenable for machine learning modeling. The proposed approach being cost-effective can be used widely for the creation of music.

vii

1. **INTRODUCTION**

Music is an [art form](https://en.wikipedia.org/wiki/The_arts#Music), and [cultural](https://en.wikipedia.org/wiki/Culture) activity, whose [medium](https://en.wikipedia.org/wiki/Art_medium#Sound) is sound. General [definitions of music](https://en.wikipedia.org/wiki/Definitions_of_music) include common elements such as [pitch](https://en.wikipedia.org/wiki/Pitch_(music)) (which governs [melody](https://en.wikipedia.org/wiki/Melody) and [harmony](https://en.wikipedia.org/wiki/Harmony)), [rhythm](https://en.wikipedia.org/wiki/Rhythm) (and its associated concepts [tempo](https://en.wikipedia.org/wiki/Tempo), [meter](https://en.wikipedia.org/wiki/Meter_(music)), and [articulation](https://en.wikipedia.org/wiki/Articulation_(music))), [dynamics](https://en.wikipedia.org/wiki/Dynamics_(music)) (loudness and softness), and the sonic qualities of [timbre](https://en.wikipedia.org/wiki/Timbre) and [texture](https://en.wikipedia.org/wiki/Texture_(music)) (which are sometimes termed the "color" of a musical sound). Different [styles or types](https://en.wikipedia.org/wiki/Music_genre) of music may emphasize, de-emphasize or omit some of these elements. Music is performed with a vast range of [instruments](https://en.wikipedia.org/wiki/Musical_instrument) and vocal techniques ranging from singing to [rapping](https://en.wikipedia.org/wiki/Rapping); there are solely [instrumental pieces](https://en.wikipedia.org/wiki/Instrumental_music), [solely vocal pieces](https://en.wikipedia.org/wiki/A_capella) (such as songs without instrumental [accompaniment](https://en.wikipedia.org/wiki/Accompaniment)) and pieces that combine singing and instruments.

The creation, performance, significance, and even the definition of music varies according to culture and social context. Indeed, throughout history, some new forms or styles of music have been criticized as "not being music", including [Beethoven](https://en.wikipedia.org/wiki/Ludwig_van_Beethoven)'s [Grosse Fuge](https://en.wikipedia.org/wiki/Grosse_Fuge) [string quartet](https://en.wikipedia.org/wiki/String_quartet) in 1825,early [jazz](https://en.wikipedia.org/wiki/Jazz) in the beginning of the 1900sand [hardcore punk](https://en.wikipedia.org/wiki/Hardcore_punk) in the 1980s[2]. There are many types of music, including [popular music](https://en.wikipedia.org/wiki/Popular_music), [traditional music](https://en.wikipedia.org/wiki/Traditional_music), [art music](https://en.wikipedia.org/wiki/Art_music), [music written for religious ceremonies](https://en.wikipedia.org/wiki/Liturgical_music) and [work songs](https://en.wikipedia.org/wiki/Work_song) such as [chanteys](https://en.wikipedia.org/wiki/Chantey). Music ranges from strictly organized compositions—such as Classical music symphonies from the 1700s and 1800s—through to spontaneously played [improvisational music](https://en.wikipedia.org/wiki/Musical_improvisation) such as [jazz](https://en.wikipedia.org/wiki/Jazz), and [avant-garde](https://en.wikipedia.org/wiki/Avant-garde) styles of [chance-based](https://en.wikipedia.org/wiki/Aleatory) [contemporary music](https://en.wikipedia.org/wiki/Contemporary_music) from the 20th and 21st centuries.[1]

Music can be divided into [genres](https://en.wikipedia.org/wiki/Music_genre) (e.g., [country music](https://en.wikipedia.org/wiki/Country_music)) and genres can be further divided into [subgenres](https://en.wikipedia.org/wiki/Subgenre) (e.g., [country blues](https://en.wikipedia.org/wiki/Country_blues) and [pop country](https://en.wikipedia.org/wiki/Pop_country) are two of the many country subgenres), although the dividing lines and relationships between music genres are often subtle, sometimes open to personal interpretation, and occasionally controversial. For example, it can be hard to draw the line between some early 1980s [hard rock](https://en.wikipedia.org/wiki/Hard_rock) and [heavy metal](https://en.wikipedia.org/wiki/Heavy_metal_music). Within [the arts](https://en.wikipedia.org/wiki/The_arts), music may be classified as a [performing art](https://en.wikipedia.org/wiki/Performing_arts), a fine art or as an auditory art. Music may be played or sung and heard live at a [rock concert](https://en.wikipedia.org/wiki/Rock_concert) or orchestra performance, heard live as part of a [dramatic work](https://en.wikipedia.org/wiki/Theatre_music) (a [music theater](https://en.wikipedia.org/wiki/Music_theater) show or opera), or it may be recorded and listened to on a radio, MP3 player, [CD player](https://en.wikipedia.org/wiki/CD_player), [smartphone](https://en.wikipedia.org/wiki/Smartphone) or as [film score](https://en.wikipedia.org/wiki/Film_score) or TV show.

Fig:1.1 Music Generation in the past Fig:1.2 Music Generation in the contemporary

## 1.1-Motivation:

## Motivating and inspirational music can feel and make us happy, emotional,and moving. People use motivational and inspirational music for gym-workouts, studying, sports, speeches, life lessons, work and more. This genre of music sometimes also goes well with movies, short films, etc. Hence there is need for music and that fascinated me to take on this project.

## 1.2-Problem Statement:

Music Generation is a complex task and is requires a great talent. To bring down the complexity in composing a good music this project can be used. It can be helpful to music composers and melophiles to generate new music from the available ones. It is also used to recreate similar music of those in the past. There has been significant interest in computational music generation for many decades [4,5]. More recently, deep learning and modern generative modelling techniques have been applied to this task [3,5]. Although music can be represented as a waveform, we can represent it more concisely by abstracting away the idiosyncrasies of a particular performance. Furthermore, symbolic representations are often tailored to instruments, which reduces their generality and implies that a lot of work is required to apply existing modelling techniques to new instruments.

**1.3-Proposed System:**

. The proposed system is of Deep-learning approach which feeds in extra information about the targets into the model to aid the prediction process, the observation that the melody is relatively independent of the starting note, and the difference in pitches of notes from the start note preserves the characteristics of a song in a compact manner, amenable for machine learning modeling. The proposed approach being cost-effective can be used widely for the creation of Jazz Music which is the main theme of the proposed model.

**Advantages of the proposed system**:

* Cost-effective.
* Tempering of the music is serene.
* Can be generated with ease.

## 1.4-Requirements Specification:

### 1.4.1- Hardware Requirements:

* System: Windows 7 or Higher.
* Hard Disk:120GB
* RAM:8GB (min)
* CPU Speed: 2.30 GHz.
* Processor: Intel Core i5.
* Input Devices: Keyboard, Mouse, Display screen

### 1.4.2- Software Requirements:

* Language: Python 3.6 or higher.
* Tool: Anaconda Navigator (Jupyter Notebook)
* Packages: Music21, Musicgen, Play, Tqdm.
* Dataset: Midi files.

# 2. LITERATURE SURVEY

### S. Hoch Reiter, J. Schmid Huber:

### "Long short-term memory", Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997.

### Sequence-based models such as Long-Short Term Memory (LSTM) [5] models and Recurrent Neural Networks (RNNs) are most commonly used for these type of problems as they have shown promising results in learning sequence information from time series data [1] [2].

LSTMs[1] are special type RECCURENT NEURAL NETWORK which process longer sequences.

### A. Graves:

### "Generating sequences with recurrent neural networks", 2013.

This paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). It is then extended to handwriting synthesis by allowing the network to condition its predictions on a text sequence. The resulting system is able to generate highly realistic cursive handwriting in a wide variety of styles.

### K. Choi, G. Fazekas, M. Sandler

### "Text-based LSTM networks for automatic music composition", 2016.

### Sequence-based models such as Long-Short Term Memory (LSTM) [1] models and Recurrent Neural Networks (RNNs) are most commonly used for these type of problems as they have shown promising results in learning sequence information from time series data [1] [3].

### [3] have explored applications of character and word-based RNNs with LSTM units for automatic generation of jazz chord progressions and rock music drum tracks.

### [Aaron van den Oord](https://arxiv.org/search/cs?searchtype=author&query=van+den+Oord%2C+A), [Sander Dieleman](https://arxiv.org/search/cs?searchtype=author&query=Dieleman%2C+S), [Heiga Zen](https://arxiv.org/search/cs?searchtype=author&query=Zen%2C+H), [Karen Simonyan](https://arxiv.org/search/cs?searchtype=author&query=Simonyan%2C+K), [Oriol Vinyals](https://arxiv.org/search/cs?searchtype=author&query=Vinyals%2C+O), [Alex Graves](https://arxiv.org/search/cs?searchtype=author&query=Graves%2C+A), [NalKalchbrenner](https://arxiv.org/search/cs?searchtype=author&query=Kalchbrenner%2C+N), [Andrew Senior](https://arxiv.org/search/cs?searchtype=author&query=Senior%2C+A), [KorayKavukcuoglu](https://arxiv.org/search/cs?searchtype=author&query=Kavukcuoglu%2C+K):

### "Wavenet: A generative model for raw audio", 2016.

This paper introduces WaveNet, a deep neural network for generating raw audio waveforms. The model is fully probabilistic and autoregressive, with the predictive distribution for each audio sample conditioned on all previous ones; nonetheless we show that it can be efficiently trained on data with tens of thousands of samples per second of audio. When applied to text-to-speech, it yields state-of-the-art performance, with human listeners rating it as significantly more natural sounding than the best parametric and concatenative systems for both English and Mandarin. A single WaveNet can capture the characteristics of many different speakers with equal fidelity, and can switch between them by conditioning on the speaker identity. When trained to model music, we find that it generates novel and often highly realistic musical fragments. We also show that it can be employed as a discriminative model, returning promising results for phoneme recognition.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Remarks** | 1. Helps to build recurrent neural networks. 2. Infers the importance of LSTM in continuous note sequences. | LSTM networks for automatic composition and reported results for generating chord progressions and rock drum tracks. Word-RNNs showed good results in both cases while char-RNNs only successfully learned chord progressions. | The ability of Long Short-Term Memory recurrent neural networks to generate both discrete and real-valued sequences with complex, long-range structure using next-step prediction. | A deep generative model of audio data that operates directly at the waveform level. Wave Nets are autoregressive and combine causal filters with dilated convolutions to allow their receptive fields to grow exponentially with depth, which is important to model the long-range temporal dependencies in audio signals. |
| **Methodology** | Gradient Based Method, Recurrent Networks Algorithms | Markov chains for automatic composition | Fuzzy Logic, Recurrent Neural Networks | \  Pixel CNN,  Text-to-speech,  Wave net |
| **Year** | 1997 | 2016 | 2013 | **2016** |
| **Authors** | S. Hoch Reiter, J. Schmid Huber: | K. Choi, G. Fazekas, M. Sandler | A. Graves: | [Aaron van den Oord](https://arxiv.org/search/cs?searchtype=author&query=van+den+Oord%2C+A), [Sander Dieleman](https://arxiv.org/search/cs?searchtype=author&query=Dieleman%2C+S), [Heiga Zen](https://arxiv.org/search/cs?searchtype=author&query=Zen%2C+H), [Karen Simonyan](https://arxiv.org/search/cs?searchtype=author&query=Simonyan%2C+K), [Oriol Vinyals](https://arxiv.org/search/cs?searchtype=author&query=Vinyals%2C+O), [Alex Graves](https://arxiv.org/search/cs?searchtype=author&query=Graves%2C+A), [NalKalchbrenner](https://arxiv.org/search/cs?searchtype=author&query=Kalchbrenner%2C+N), [Andrew Senior](https://arxiv.org/search/cs?searchtype=author&query=Senior%2C+A), [KorayKavukcuoglu](https://arxiv.org/search/cs?searchtype=author&query=Kavukcuoglu%2C+K): |
| **Title** | "Long short-term memory" | "Text-based LSTM networks for automatic music composition” | "Generating sequences with recurrent neural networks" | "Wave net:  A generative model for raw audio” |
| **SNo** | 1 | 2 | 3 | 4 |

**Table 2.1:** Literature Survey

# 3.Music Generation Stratagem

## 3.1-System Architecture:

Data Preproces-sing

Dataset

Conversion of pitches in the notes in string format to integer format

Creation Of Note\_sequences

Creation of array of input and output sequences to train the model

Re-shaping and

normalizing

the input

vector sequence

Midi file is created and can be played using play.midi()

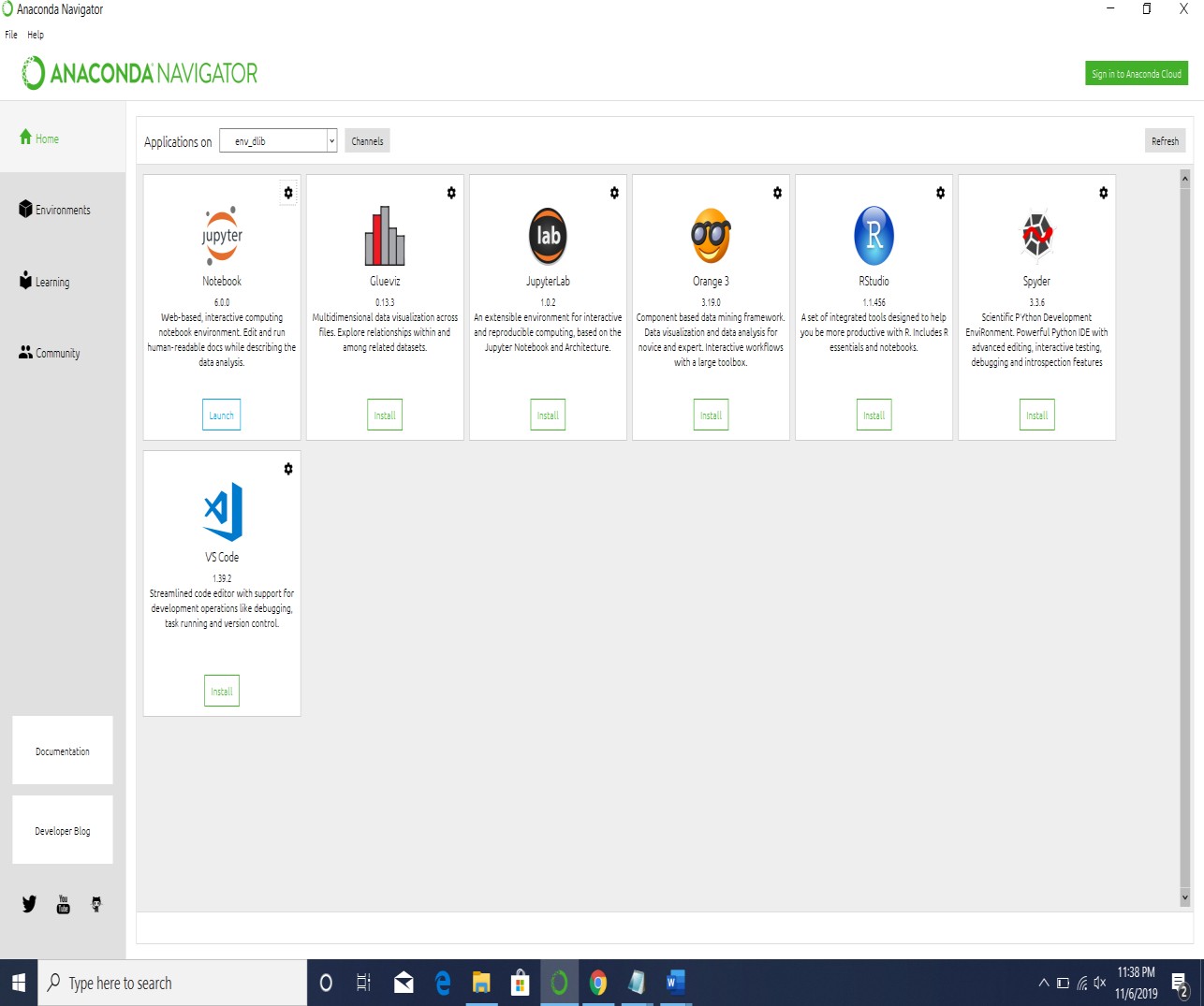
Generate the music using the model

Train the neural network using the preprocessed data

**Fig: 3.1** Architecture of The Neural Network Model Designed

### 3.2- Tools

**Anaconda Navigator:**

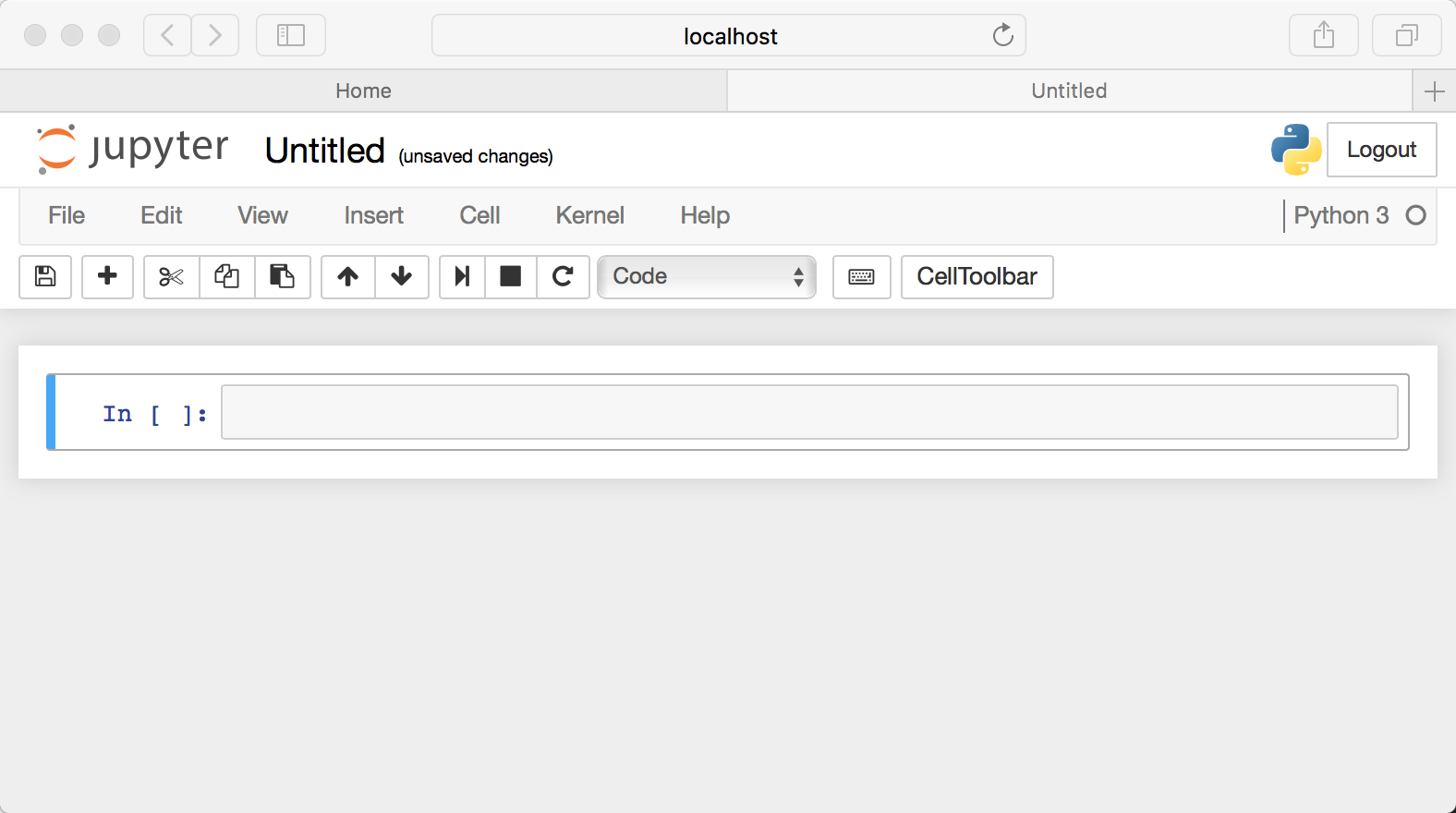
Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

**Fig 3.2** : Anaconda Navigator

As shown in above figure1.1, the following applications are available by default in Navigator: [Jupyter Lab,](https://en.wikipedia.org/wiki/Project_Jupyter#Jupyter_Lab) [Jupyter-Notebook](https://en.wikipedia.org/wiki/Project_Jupyter#Jupyter_Notebook), [Qt Console,](https://qtconsole.readthedocs.io/en/latest/)  [Orange](https://en.wikipedia.org/wiki/Orange_(software)), [R-studio](https://en.wikipedia.org/wiki/Rstudio), [VisualStudioCode](https://en.wikipedia.org/wiki/Visual_Studio_Code).

### Jupyter Notebook

As shown in figure 1.2, the Jupyter [Notebook](https://en.wikipedia.org/wiki/Notebook_interface) (formerly IPython Notebooks) is a [web-based](https://en.wikipedia.org/wiki/Rich_Internet_application) [interactive](https://en.wikipedia.org/wiki/Rich_Internet_application) computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python webserver, or Jupyter document format depending on context. A Jupyter Notebook document is a [JSON](https://en.wikipedia.org/wiki/JSON) document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using [Markdown](https://en.wikipedia.org/wiki/Markdown)), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

A Jupyter Notebook can be converted to a number of [open standard](https://en.wikipedia.org/wiki/Open_standard) output formats ([HTML,](https://en.wikipedia.org/wiki/HTML) [presentation slides,](https://en.wikipedia.org/wiki/Presentation_slide) [LaTeX,](https://en.wikipedia.org/wiki/LaTeX) [PDF, Re-Structured Text,](https://en.wikipedia.org/wiki/PDF) [Markdown](https://en.wikipedia.org/wiki/Markdown), [Python](https://en.wikipedia.org/wiki/Python_(programming_language))) through "Download As" in the web interface, via the [nbconvert](https://nbconvert.readthedocs.io/) library or "jupyternb convert" command line interface in a shell.

**Fig:3.3**: Jupyter Notebook

## 3.3- Machine Learning Models:

### 3.3.1- Music 21 Module:

Music21 is a Python-based toolkit for computer-aided musicology. People use music21 to answer questions from musicology using computers, to study large datasets of music, to generate musical examples, to teach fundamentals of music theory, to edit musical notation, study music and the brain, and to compose music (both algorithmically and directly).One of music21’s mottos is “Listen Faster.” With the toolkit you should be able to find interesting moments and get a sense of the overall profile of a piece or a repertory of pieces. We hope that with the computer you’ll have more time for listening and playing for enjoyment and use less of your time listening for work.

The system has been around since 2008 and is constantly growing and expanding. The approaches and traditions in music21 have been used in many previous software systems. See [Authors, Acknowledgments, Contributing, and Licensing](https://web.mit.edu/music21/doc/about/about.html#about) for information on the authors and background of the project. The 21 in music21 refers to its origins as a project nurtured at MIT. At MIT all courses have numbers and music, along with some other humanities departments, are numbered 21. The music departments of MIT, along with Harvard, Smith, and Mount Holyoke Colleges, helped bring this toolkit from its easiest roots to a mature system.

A screenshot of a computer screen

Description automatically generated

**Fig:3.4** Chord Sequences Generated using Music 21 Library

**3.3.2-Pickle Library:**

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

Advantages of using Pickle Module:

1. Recursive objects (objects containing references to themselves): Pickle keeps track of the objects it has already serialized, so later references to the same object won’t be serialized again. (The marshal module breaks for this.)
2. Object sharing (references to the same object in different places): This is similar to self- referencing objects; pickle stores the object once,and ensures that all other references point to the master copy. Shared objects remain shared, which can be very important for mutable objects.
3. User-defined classes and their instances: Marshal does not support these at all, but pickle can save and restore class instances transparently. The class definition must be importable and live in the same module as when the object was stored.

**3.3.3-Keras Module:**

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

1. Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

2. Supports both convolutional networks and recurrent networks, as well as combinations of the two.

3. Runs seamlessly on CPU and GPU.

# 3.4 Dataset:

Our dataset includes piano tunes stored in the **MIDI** format. MIDI (Musical Instrument Digital Interface) is a protocol which allows electronic instruments and other digital musical tools to communicate with each other. Since a MIDI file only represents player information, i.e., a series of messages like ‘note on’, ‘note off, it is more compact, easy to modify, and can be adapted to any instrument.

Before we move forward, let us understand some music related terminologies:

* Note: A note is either a single sound or its representation in notation. Each note consist of pitch, octave, and an offset.
* Pitch: Pitch refers to the frequency of the sound.
* Octave: An octave is the interval between one musical pitch and another with half or double its frequency.
* Offset: Refers to the location of the note.
* Chord: Playing multiple notes at the same time constitutes a chord.

A screenshot of a computer

Description automatically generated

Fig:3.5 Dataset representing the MIDI files

## ****3.5 Data Preprocessing:****

We will use the music21 [toolkit](http://web.mit.edu/music21/doc/moduleReference/moduleConverter.html) (a toolkit for computer-aided musicology, MIT) to extract data from these MIDI files.

From Batch Dataset notes are extracted

The function get\_notes returns a list of notes and chords present in the .mid file. We use the converter.parse() function to convert the midi file in a stream object, which in turn is used to extract notes and chords present in the file. The list returned by the function get\_notes() looks as follows:

**A screenshot of a social media post

Description automatically generated**

### Fig:3.6 Notes Extraction from the input data

|  |  |
| --- | --- |
|  | Out:  ['F2', '4.5.7', '9.0', 'C3', '5.7.9', '7.0', 'E4', '4.5.8', '4.8', '4.8', '4', 'G#3',  'D4', 'G#3', 'C4', '4', 'B3', 'A2', 'E3', 'A3', '0.4', 'D4', '7.11', 'E3', '0.4.7', 'B4', 'C3', 'G3', 'C4', '4.7', '11.2', 'C3', 'C4', '11.2.4', 'G4', 'F2', 'C3', '0.5', '9.0', '4.7', 'F2', '4.5.7.9.0', '4.8', 'F4', '4', '4.8', '2.4', 'G#3',  '8.0', 'E2', 'E3', 'B3', 'A2', '4.9', '0.4', '7.11', 'A2', '9.0.4', ...........]  We can see that the list consists of pitches and chords (represented as a list of integers separated by a dot). We assume each new chord to be a new pitch on the list. As letters are used to generate words in a sentence, similarly the music vocabulary used to generate music is defined by the unique pitches in the notes list. |

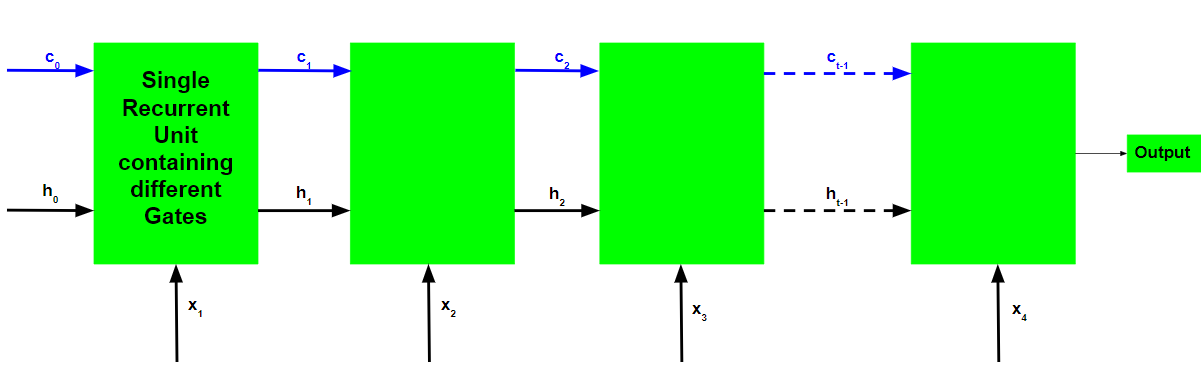
## 3.5- Algorithm:

## We use Long Short-Term Memory (LSTMs) a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was desgined by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give efficent performance. LSTM can by default retain the information for long period of time. It is used for processing, predicting and classifying on the basis of time series data.

In concept, an LSTM recurrent unit tries to “remember” all the past knowledge that the network is seen so far and to “forget” irrelevant data. This is done by introducing different activation function layers called “gates” for different purposes. Each LSTM recurrent unit also maintains a vector called the **Internal Cell State** which conceptually describes the information that was chosen to be retained by the previous LSTM recurrent unit. A Long Short Term Memory Network consists of four different gates for different purposes as described below:-

1. **Forget Gate(f):** It determines to what extent to forget the previous data.
2. **Input Gate(i):** It determines the extent of information to be written onto the Internal Cell State.
3. **Input Modulation Gate(g):** It is often considered as a sub-part of the input gate and many literatures on LSTM’s do not even mention it and assume it inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information **Zero-mean**. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate’s actions are less important than the others and is often treated as a finesse-providing concept, it is good practice to include this gate into the structure of the LSTM unit.
4. **Output Gate(o):** It determines what output(next Hidden State) to generate from the current Internal Cell State.

The basic work-flow of a Long Short Term Memory Network is similar to the work-flow of a Recurrent Neural Network with only difference being that the Internal Cell State is also passed forward along with the Hidden State.



**Working of an LSTM recurrent unit:**

1. Take input the current input, the previous hidden state and the previous internal cell state.
2. Calculate the values of the four different gates by following the below

steps:-

* + For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
  + Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

**Input Gate : Sigmoid Function**

**Forget Gate : Sigmoid Function**

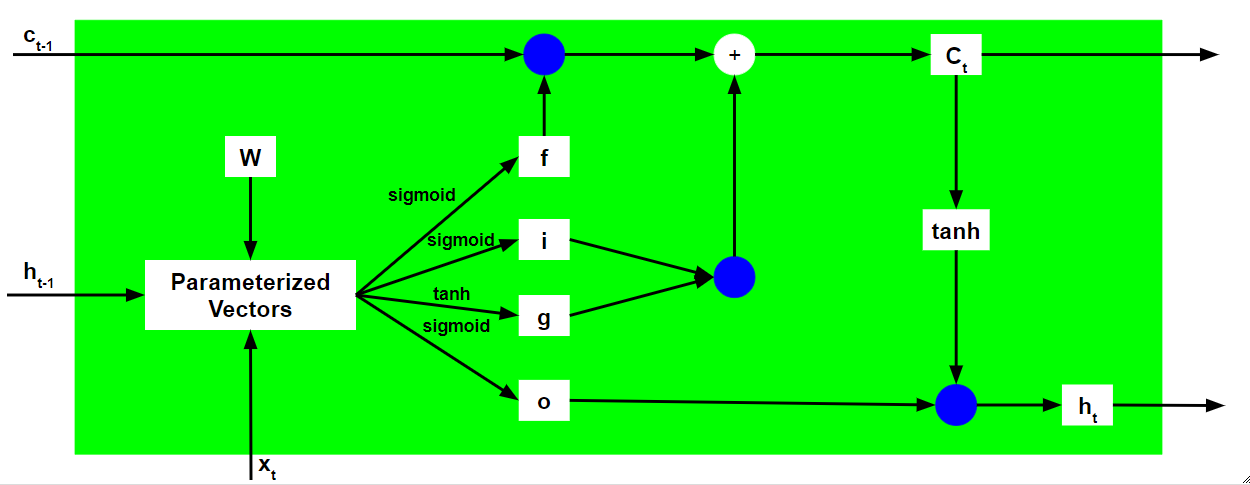
**Output Gate : Sigmoid Function**

**Input Modulation Gate : Hyperbolic Tangent Function**

1. Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then adding the two vectors.

1. Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element wise multiplication with the output gate.

The above stated working is illustrated as below:-



Note that the blue circles denote element-wise multiplication. The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, an LSTM network also generates an output at each time step and this output is used to train the network using gradient descent.



The only main difference between the Back-Propagation algorithms of Recurrent Neural Networks and Long Short Term Memory Networks is related to the mathematics of the algorithm.

Let (yt) be the predicted output at each time step and (yt) be the actual output at each time step. Then the error at each time step is given by:-

The total error is thus given by the summation of errors at all time steps.

Similarly, the value  can be calculated as the summation of the gradients at each time step.

Using the chain rule and using the fact that  is a function of   and which indeed is a function of  , the following expression arises: -

A screenshot of a cell phone

Description automatically generated

Thus the total error gradient is given by the following: -

A screenshot of a cell phone

Description automatically generated

Note that the gradient equation involves a chain of  for an LSTM Back-Propagation while the gradient equation involves a chain of   for a basic Recurrent Neural Network.

**How does LSTM solve the problem of vanishing and exploding gradients?**

Recall the expression for .

The value of the gradients is controlled by the chain of derivatives starting from  . Expanding this value using the expression for :-

A screenshot of a social media post

Description automatically generated

For a basic RNN, the term  after a certain time starts to take values either greater than 1 or less than 1 but always in the same range. This is the root cause of the vanishing and exploding gradients problem. In an LSTM, the term    does not have a fixed pattern and can take any positive value at any time step. Thus, it is not guaranteed that for an infinite number of time steps, the term will converge to 0 or diverge completely. If the gradient starts converging towards zero, then the weights of the gates can be adjusted accordingly to bring it closer to 1. Since during the training phase, the network adjusts these weights only, it thus learns when to let the gradient converge to zero and when to preserve it. In this way a LSTM is different from the recurrent neuralnetwork.

# 4.Training and Results

# 4.1 Training:

We update the weight of the model by iterating a number of music’s in the dataset and preprocess the data as stated above. Then we take several instances in a batch to be input and the target of the neural network. We will use the Sequential,LSTM an inbuilt module of the keras is imported and model is created using it. Once the model is generated then we train the network using the preloaded checkpoint that saved the best model weights. This will convert the model and train the model accordingly with a great accuracy as we constructed it with nearly 200 epochs for greater accuracy make our training a lit bit slower but yields greater composition. The function cannot use different size of batches as the input of neural network. For example, our batch size is 64. If the size of datasets is 70, the last batch will contain 6 instances. This will throw exception to the program as the graph will have an input with different size from the initial graph. Maybe it works by creating placeholders by seeing the first input when using the function. In this project we will use 12 BATCH\_SONG and 96 BATCH\_NNET\_SIZE. That means we will take 12 music’s from the list of all music’s, then extract its sequence. Then for every step in the neural network, we take 200 epochs for construction of the model sequences from the extracted sequence instances to be input and target of neural network. A screenshot of a social media post

Description automatically generated

Fig:4.1 Training the Model

The train\_ network () function in return calls the get\_notes () where the notes after preprocessing are stored as pitches in the form of strings. But, the neural network only takes integer as input hence the pitches in the form of string are enumerated to integers and stored in the input ouput\_sequences.

A screenshot of a computer

Description automatically generated

Fig:4.2 Creation of Model

As the model is generated now it is essential to generate the note\_sequences() out of it for the creation of new midi file. So, we use generate() method inorder to create the midi file.

A screenshot of a computer

Description automatically generated

Fig:4.3 Generation of a piano midi file

## 

## 4.2-Results:

## The results play a key role in understanding the functioning of the model and estimating its accuracy. So, better and efficient results help in understanding the model even better.

## A screenshot of a social media post Description automatically generated

**Fig 4.4:** The code showing conversion of the predicted output into midi format so that it is audible.

Finally generating the new music file out of the available music sequences.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

**Figure 4.5: test\_output file is created to hear the created music sequence.**

* **We can use the model to create music of a specific genre Artificially. Which helps in recreating the music that even existed in the past.**
* **In this we used the model for creating jazz genre music from the previously existing music**

Western Jazz music skies album

****

Rehman Songs

Alan-walker music

****

# 5.CONCLUSION AND FUTURE SCOPE

## In this project, we introduced LSTM network with the goal of generating jazz music on piano. In general, our model improved the quality of generated music through learning context information of notes from the input sequence fed and developed a model by rigorously training the network. We redesign the train\_network function to avoid generating a lot of meaningless results, which accelerate model optimization process. Through input chord information of the measure during the training stage, our model permit user custom input chord to control music chord progression, which is very meaningful for the composer. However, music generated by our model lack of long-term structure same as the most existing systems, which are still a problem yet to be solved. With the assistance of experts in the field of music, a detailed strategy may be manually designed to control the generation process of the model with the goal of having a long-term structure. In addition, like a language model, embedding representation of note may be helpful for generating melody, which is worth researching in the future.

# BIBLIOGRAPHY

### [Kozinn, Allen](https://en.wikipedia.org/wiki/Allan_Kozinn): ["John Cage, 79, a Minimalist Enchanted with Sound, Dies"](https://query.nytimes.com/gst/fullpage.html?res=9E0CE1DB1E3BF930A2575BC0A964958260). [The New York Times](https://en.wikipedia.org/wiki/The_New_York_Times). Retrieved 11 September 2012.

### Manabe, Noriko: The Revolution Will Not Be Televised: Protest Music After Fukushima. Oxford University Press, 2015. p. 163.

### S. Hoch Reiter, J. Schmid Huber: “Long short-term memory", Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997.

### K. Choi, g. Fazekas, M. Sandler: "text-based Long short term networks for automatic music composition", 2016.

### A. Graves: "Generating sequences with recurrent neural networks", 2013

### [Aaron van den Oord](https://arxiv.org/search/cs?searchtype=author&query=van+den+Oord%2C+A), [Sander Dieleman](https://arxiv.org/search/cs?searchtype=author&query=Dieleman%2C+S), [Heiga Zen](https://arxiv.org/search/cs?searchtype=author&query=Zen%2C+H), [Karen Simonyan](https://arxiv.org/search/cs?searchtype=author&query=Simonyan%2C+K), [Oriol Vinyals](https://arxiv.org/search/cs?searchtype=author&query=Vinyals%2C+O), [Alex Graves](https://arxiv.org/search/cs?searchtype=author&query=Graves%2C+A), [NalKalchbrenner](https://arxiv.org/search/cs?searchtype=author&query=Kalchbrenner%2C+N), [Andrew Senior](https://arxiv.org/search/cs?searchtype=author&query=Senior%2C+A), KorayKavukcuoglu: “Wave net: A generative model for raw audio", 2016.

# 

# APPENDIX

## /\*Music Generation Using Deep-Learning\*/

### /\* Import libraries and modules \*/

import Pygame

**/\*Defining the required function to play the file created\*/**

def play\_music(music\_file):

**"""**

**stream music with mixer.music module in blocking manner**

**this will stream the sound from disk while playing**

**"""**

clock = pygame.time.Clock()

try:

pygame.mixer.music.play(music\_file)

print ("Music file %s loaded!" % music\_file)

except pygame.error:

print ("File %s not found! (%s)" % (music\_file, pygame.get\_error ()))

return

pygame.mixer.music.play()

while pygame.mixer.music.get\_busy():

# check if playback has finished

clock.tick(30)

**/\*pick a midi music file you have ...\*/**

**/\* (if not in working folder use full path) \*/**

def play\_midi(midi\_file):

freq = 44100 # audio CD quality

bitsize = -16 # unsigned 16 bit

channels = 2 # 1 is mono, 2 is stereo

buffer = 1024 # number of samples

pygame.mixer.init(freq, bitsize, channels, buffer)

**/\* optional volume 0 to 1.0\*/**

pygame.mixer.music.set\_volume(0.8)

try:

play\_music(midi\_file)

except Keyboard Interrupt:

# if user hits Ctrl/C then exit

# (works only in console mode)

pygame.mixer.music.fadeout(1000)

pygame.mixer.music.stop()

raise System Exit

**/\*Entering into the Main Code\*/**

### /\* Import libraries and modules \*/

### import sys

### import re

### import numpy as np

### import pandas as pd

### import music21

### from glob import glob

### import IPython

### from tqdm import tqdm

### import pickle

### from keras.utils import np\_utils

### import play

### from music21 import converter, instrument, note, chord, stream

### /\*Loading the music files\*/

### songs = glob('Jazz/\*.mid')

### songs = songs [:12]

### /\*Displaying all the songs that’s been chosen in batch\*/

### print(songs)

**/\*Generating the notes out of the loaded music files\*/**

def get\_notes():

notes = []

for file in songs:

# converting .mid file to stream object

midi = converter.parse(file)

notes\_to\_parse = []

try:

# Given a single stream, partition into a part for each unique instrument

parts = instrument.partitionByInstrument(midi)

except:

pass

if parts: # if parts has instrument parts

notes\_to\_parse = parts.parts[0].recurse()

else:

notes\_to\_parse = midi.flat.notes

for element in notes\_to\_parse:

if isinstance(element, note.Note):

# if element is a note, extract pitch

notes.append(str(element.pitch))

elif(isinstance(element, chord.Chord)):

# if element is a chord, append the normal form of the

# chord (a list of integers) to the list of notes.

notes.append('.'.join(str(n) for n in element.normalOrder))

with open('data/notes', 'wb') as filepath:

pickle.dump(notes, filepath)

return notes

**/\*Preparing note sequences from the obtained notes\*/**

def prepare\_sequences(notes, n\_vocab):

sequence\_length = 100

**/\* Extract the unique pitches in the list of notes.\*/**

pitchnames = sorted(set(item for item in notes))

**/\*Create a dictionary to map pitches to integers\*/**

note\_to\_int = dict((note, number) for number, note in enumerate(pitchnames))

network\_input = []

network\_output = []

/\***create input sequences and the corresponding outputs\*/**

for i in range(0, len(notes) - sequence\_length, 1):

sequence\_in = notes[i: i + sequence\_length]

sequence\_out = notes[i + sequence\_length]

network\_input.append([note\_to\_int[char] for char in sequence\_in])

network\_output.append(note\_to\_int[sequence\_out])

n\_patterns = len(network\_input)

**/\*Reshape the input into a format compatible with LSTM layers \*/**

network\_input = np.reshape(network\_input, (n\_patterns, sequence\_length, 1))

**/\* Normalize input \*/**

network\_input = network\_input / float(n\_vocab)

**/\* One hot encode the output vectors \*/**

network\_output = np\_utils.to\_categorical(network\_output)

return (network\_input, network\_output)

/\***Generating the models from the keras module \*/**

from keras.models import Sequential

from keras.layers import Activation, Dense, LSTM, Dropout, Flatten

def create\_network(network\_in, n\_vocab):

**"""Create the model architecture"""**

model = Sequential()

model.add(LSTM(128,input\_shape=network\_in.shape[1:], return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(128, return\_sequences=True))

model.add(Flatten())

model.add(Dense(256))

model.add(Dropout(0.3))

model.add(Dense(n\_vocab))

model.add(Activation('softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam')

return model

**/\*Importing the model checkpoint\*/**

from keras.callbacks import ModelCheckpoint

def train(model, network\_input, network\_output, epochs):

**"""**

**Train the neural network**

**"""**

# Create checkpoint to save the best model weights.

filepath = 'weights.best.music3.hdf5'

checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=0, save\_best\_only=True)

model.fit(network\_input,network\_output,epochs=epochs,batch\_size=12, callbacks=[checkpoint])

**/\*Training the module with the loaded dataset\*/**

def train\_network():

**"""**

**Get notes**

**Generates input and output sequences**

**Creates a model**

**Trains the model for the given epochs**

**"""**

epochs = 2

notes = get\_notes()

print('Notes processed')

n\_vocab = len(set(notes))

print('Vocab generated')

network\_in, network\_out = prepare\_sequences(notes, n\_vocab)

print('Input and Output processed')

model = create\_network(network\_in, n\_vocab)

print('Model created')

return model

print('Training in progress')

train(model, network\_in, network\_out, epochs)

print('Training completed')

### Train the model

train\_network()

**/\*Generating the sequences from the frequent patterns identified in the training phase\*/**

def generate():

""" Generate a piano midi file """

#load the notes used to train the model

with open('data/notes', 'rb') as filepath:

notes = pickle.load(filepath)

# Get all pitch names

pitchnames = sorted(set(item for item in notes))

# Get all pitch names

n\_vocab = len(set(notes))

print('Initiating music generation process.......')

network\_input = get\_inputSequences(notes, pitchnames, n\_vocab)

normalized\_input = network\_input / float(n\_vocab)

model = create\_network(normalized\_input, n\_vocab)

print('Loading Model weights.....')

model.load\_weights('weights.best.music3.hdf5')

print('Model Loaded')

prediction\_output = generate\_notes(model, network\_input, pitchnames, n\_vocab)

create\_midi(prediction\_output)

def get\_inputSequences(notes, pitchnames, n\_vocab):

""" Prepare the sequences used by the Neural Network """

# map between notes and integers and back

note\_to\_int = dict((note, number) for number, note in enumerate(pitchnames))

sequence\_length = 100

network\_input = []

for i in range(0, len(notes) - sequence\_length, 1):

sequence\_in = notes[i:i + sequence\_length]

network\_input.append([note\_to\_int[char] for char in sequence\_in])

network\_input = np.reshape(network\_input, (len(network\_input), 100, 1))

return (network\_input)

def generate\_notes(model, network\_input, pitchnames, n\_vocab):

""" Generate notes from the neural network based on a sequence of notes """

**# Pick a random integer**

start = np.random.randint(0, len(network\_input)-1)

int\_to\_note = dict((number, note) for number, note in enumerate(pitchnames))

# pick a random sequence from the input as a starting point for the prediction

pattern = list(network\_input[start])

prediction\_output = []

print('Generating notes........')

# generate 500 notes

for note\_index in range(500):

prediction\_input = np.reshape(pattern, (1, len(pattern), 1))

prediction\_input = prediction\_input / float(n\_vocab)

prediction = model.predict(prediction\_input, verbose=0)

# Predicted output is the argmax(P(h|D))

index = np.argmax(prediction)

# Mapping the predicted interger back to the corresponding note

result = int\_to\_note[index]

# Storing the predicted output

prediction\_output.append(result)

pattern.append(index)

# Next input to the model

pattern = pattern[1:len(pattern)]

print('Notes Generated...')

return prediction\_output

**/\*Creating midi file out of the predicted note sequences\*/**

def create\_midi(prediction\_output):

""" convert the output from the prediction to notes and create a midi file

from the notes """

offset = 0

output\_notes = []

# create note and chord objects based on the values generated by the model

for pattern in prediction\_output:

# pattern is a chord

if ('.' in pattern) or pattern.isdigit():

notes\_in\_chord = pattern.split('.')

notes = []

for current\_note in notes\_in\_chord:

new\_note = note.Note(int(current\_note))

new\_note.storedInstrument = instrument.Piano()

notes.append(new\_note)

new\_chord = chord.Chord(notes)

new\_chord.offset = offset

output\_notes.append(new\_chord)

# pattern is a note

else:

new\_note = note.Note(pattern)

new\_note.offset = offset

new\_note.storedInstrument = instrument.Piano()

output\_notes.append(new\_note)

# increase offset each iteration so that notes do not stack

offset += 0.5

midi\_stream = stream.Stream(output\_notes)

print('Saving Output file as midi....')

midi\_stream.write('midi', fp='test\_output4.mid')

**/\*Generate a new jazz music\*/**

generate()

**/\*Play the Jazz music\*/**

play.play\_midi('test\_output4.mid')