Algorithmic music composition using RNN

Thesis submitted in partial fulfilment

of the requirements of the degree of

Masters

in

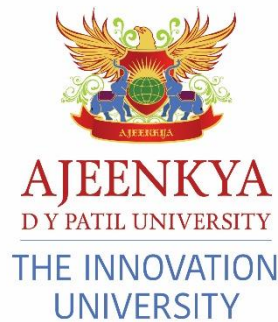**Computer Applications**

by

Anurag Kumar

2021-M-03012000

and

Yogesh Yadav

2021-M-28012000

Under the Supervision of

Dr. Anupriya Kamble

April 2023

School of Engineering

Ajeenkya D Y Patil University, Pune

CERTIFICATE

This is to certify that the dissertation entitled **"Algorithmic music composition using RNN"** is a bonafide work of "**Anurag Kumar" (2021-M-03012000),** **"Yogesh Yadav" (2021-M-28012000**) submitted to the School of Engineering, Ajeenkya D Y Patil University, Pune in partial fulfillment of the requirement for the award of the degree of **"Masters in Computer Applications"**.

Dr. Anupriya Kamble

Supervisor

Internal-Examiner

External Examiner

Dr. Biswajeet Champaty

Head-School of Engineering

Supervisor's Certificate

This is to certify that the dissertation entitled **"Algorithmic music composition using RNN"** submitted by **Anurag Kumar, URN 2021-M-03012000,** is a record of original work carried out by him/her under my supervision and guidance in partial fulfillment of the requirements of the degree of **Masters in Computer Applications** at **School of Engineering**, **Ajeenkya DY Patil University, Pune, Maharashtra-412105**. Neither this dissertation nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

_____

Dr. Anupriya Kamble

Supervisor

Declaration of Originality

I, *Anurag Kumar*, URN *2021-M-03012000* and *Yogesh Yadav*, URN *2021-M-28012000*, hereby declare that this dissertation entitled "*Algorithmic music composition using RNN*" presents my original work carried out as a master student of School of Engineering, Ajeenkya DY Patil University, Pune, Maharashtra. To the best of my knowledge, this dissertation contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of Ajeenkya D Y Patil University, Pune or any other institution. Any contribution made to this research by others, with whom I have worked at Ajeenkya DY Patil University, Pune or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission.

I am fully aware that in case of any non-compliance detected in future, the Academic Council of Ajeenkya D Y Patil University, Pune may withdraw the degree awarded to me on the basis of the present dissertation.

Date:
Place: Lohegaon, Pune

_____   _____

Anurag Kumar   **Yogesh Yadav**

Acknowledgement

I remain immensely obliged to **Dr. Anupriya Kamble**, for providing me with the idea of this topic, and for his invaluable support in garnering resources for me either by way of information or computers also his guidance and supervision which made this Internship/Project happen.

I would like to express my sincerest gratitude to Professor Ravi Khatri for their invaluable support and guidance throughout the course of this project. Their expertise in Machine learning and their willingness to share their knowledge with me have been instrumental in the success of this project. Their insights and feedback were instrumental in helping me refine my ideas and approach to the project, and their encouragement kept me motivated when the going got tough. I am incredibly fortunate to have had the opportunity to work with such a dedicated and supportive mentor. Thank you, Professor Ravi Khatri, for your contributions to this project and for being an inspiration to me as a student and aspiring Deep learning and neural network professional.

I would like to say that it has indeed been a fulfilling experience for working out this Internship/Project.

Anurag Kumar   **Yogesh Yadav**

Abstract

Deep learning advancements have led to the rising utilization of neural networks in diverse artistic domains like music, literature, and visual arts, approaching a level of performance comparable to human abilities. This paper proposes a music generation model based on bidirectional recurrent neural network, which can effectively explore the complex relationship between notes and obtain the conditional probability from time and pitch dimensions. The existing system usually ignored the information in the negative time direction, however which is non-trivial in the music prediction task, so we propose a bidirectional LSTM model to generate the note sequence. Experiments with classical piano datasets have demonstrated that we achieve high performance in music generation tasks compared to the existing unidirectional biaxial LSTM method

In the realm of music generation, numerous scholars specifically view it as a probabilistic modelling process for polyphonic music. They depict music as a sequence of notes and strive to represent it as a probability distribution. This involves assigning probabilities to the sequence of previous notes and considering contextual factors like chords and beats to determine the subsequent note. Significantly, unlike rule-based composition, this approach enables us to train a dedicated model using extensive musical corpora and permits the automatic discovery of patterns. In the generation phase, we sample from the trained probability distribution to generate new music pieces. Recurrent neural network (RNN), especially long short-term memory networks (LSTM) [3], have been shown to predict time series data effectively. In fact, many researchers have used LSTM to generate music, which has also achieved good results.

To generate music using RNNs, we first need to train the network on a dataset of existing music. This dataset can be in the form of MIDI files, which contain the note sequences of a piece of music. The RNN can then learn the patterns and structures of the music in the dataset and use this knowledge to generate new music. The RNN works by taking in a sequence of notes as input and generating a new note as output. This new note is then added to the sequence, and the process is repeated to generate a complete piece of music. The RNN can also be trained to generate music in different styles and genres by adjusting the input data. Music generation using RNNs has several applications, including music composition, film scoring, and video game music. It can also be used to create personalized music playlists based on a user's preferences and to generate background music for different types of content such as podcasts and videos.

In conclusion, music generation using RNNs is a fascinating field that has the potential to revolutionize the music industry. With the ability to generate music in different styles and genres, RNNs can be used to create original compositions that are both novel and coherent. As the technology advances, we can expect to see more sophisticated and creative applications of RNNs in music generation.

Index Terms- music generation; bidirectional recurrent neural network; deep learning

Contents

List of Figures

List of Abbreviation

LSTM- Long Short Term Memory
RNN- Recurrent Neural Network
AWS- Amazon Web Services
MIDI- Musical Instrument Digital Interface
DAW- Digital Audio Workstation
GRU- Gated Recurrent Unit
RAM- Random Access Memory
VRAM- Virtual Random Access Memory
GPU- Graphics Processing Unit
ADCs- Analog to Digital Converters
DACs- Digital to Analog Converters
CNN- Convolutional Neural Network
AIVA- Artificial Intelligence Virtual Artist
IDE- Integrated Development Environment
API- Application Programming Interface
ML- Machine Learning
AIML- Artificial Intelligence Modelling Language
ReLU- Rectified Linear Unit

CHAPTER 1

Introduction

1.1 Overview

Neural networks and machine learning are two closely related concepts that are transforming the way we solve complex problems in various domains such as computer vision, natural language processing, speech recognition, and many others.

Machine learning, a subset of artificial intelligence (AI), entails instructing machines to acquire knowledge from data without the need for direct programming. It's a method for automatically improving the performance of a system over time by using statistical algorithms and models that identify patterns in data.

Machine learning, which falls under the umbrella of artificial intelligence (AI), involves the process of enabling machines to gain knowledge from data without relying on explicit programming instructions. Neural networks consist of layers of interconnected nodes, or neurons that perform mathematical operations on incoming data and pass the results to the next layer. By adjusting the strength of connections between neurons, a neural network can learn to recognize complex patterns in data and make accurate predictions.

Neural networks have been successfully applied to a wide range of tasks, such as image classification, object detection, natural language processing, and speech recognition. They have revolutionized many industries, including healthcare, finance, transportation, and entertainment, by providing accurate and efficient solutions to complex problems.

In summary, machine learning and neural networks are powerful tools that enable us to build intelligent systems that can learn and improve from data. They have the potential to transform many aspects of our lives and create new opportunities for innovation and discovery.

Applying predictive machine learning models to original data is the goal of creative prediction. Recurrent neural networks (RNNs), which are deep learning models that may be used to produce sequential and temporal data, are the main topic of discussion. Text and music are only two examples of the creative data that may be processed with RNNs. By predicting each piece individually, they can "create" new sequences by learning the long-range structure from a corpus of data. They can be utilized for "predictive interaction" where a person works with, influences, and is influenced by a generative neural network when incorporated in a creative interface.

Deep learning has led to the growing utilization of neural networks across artistic domains like music, literature, and visual arts, to the point where their capabilities are comparable to those of humans. However, current systems tend to overlook the significance of information in the negative time direction, particularly in the task of music prediction. To address this limitation, we

introduce a bidirectional LSTM model that effectively generates note sequences. Through experiments conducted on classical piano datasets, we have successfully achieved superior performance in music generation tasks compared to the conventional unidirectional biaxial LSTM approach.

1.2 Scope of Work

The scope of work in music creation using machine learning (ML) can involve several tasks and applications, such as:

1. Music generation: ML algorithms can be used to generate new musical pieces based on a given set of rules, genres, or styles. This can be achieved through techniques such as deep learning, neural networks, and reinforcement learning.
2. Music analysis: ML can be used to analyse music pieces, identify their structure, genre, and mood, as well as detect patterns and trends in the data. This can help musicians and producers to understand the elements that make a particular song successful or popular and guide them in creating similar works[1].
3. Music recommendation: ML can be used to create personalized music recommendations for listeners based on their listening habits, preferences, and context. This can be achieved by analysing data such as user profiles, music metadata, and listening history.
4. Music transcription: ML can be used to transcribe music from audio recordings or MIDI files into sheet music or digital formats. This can be useful for musicians who want to learn and play songs or for producers who want to remix or sample existing works[1].
5. Music performance: ML can be used to enhance live music performances by automating certain tasks such as adjusting sound levels, lighting, and effects based on the performer's actions or the audience's reaction.

Overall, the scope of work in music creation using ML is vast and can involve many different applications and techniques.

1.2.1 Need of System

Making music using machine learning can be a fascinating and challenging task, but it requires some technical knowledge in both music theory and machine learning.

One popular approach to creating music with machine learning is to use a generative model, such as a neural network, to learn patterns from existing music data and then generate new pieces of music based on these patterns.

To get started with this approach, you would need a dataset of MIDI files or audio recordings of music. You could then use a neural network architecture, such as a recurrent neural network

(RNN) or a convolutional neural network (CNN), to learn the patterns in the data and generate new music.

There are also pre-built platforms and tools available, such as Magenta by Google, which provides pre-trained models and APIs for generating music using machine learning. You could also explore other software tools, such as DeepJ or AIVA, which allow you to create music using machine learning algorithms[2].

Another approach is to use machine learning to enhance or modify existing music, such as changing the tempo, adding or removing instruments, or altering the melody. For example, you could use a machine learning algorithm to create a remix of an existing song by changing the tempo, pitch, and other musical elements.

Overall, making music using machine learning requires a good understanding of music theory, programming, and machine learning concepts. It can be a fun and exciting way to explore the intersection of technology and creativity.

1.2.2 Existing Systems

There are many applications that currently use machine learning to make music, ranging from mobile apps to advanced software tools used by professional musicians and composers. Here are a few examples:

1. Amper Music: Amper is a cloud-based AI music composition platform that allows users to create original music in minutes by selecting a style, mood, and duration. It uses machine learning algorithms to generate high-quality, royalty-free music based on the user's input.
2. AIVA (Artificial Intelligence Virtual Artist): AIVA is an AI-powered composer that can create original music compositions in a variety of styles and genres. AIVA uses a deep learning algorithm to analyze existing music and learn how to compose original pieces of music.
3. Google Magenta: Google's Magenta project is an open-source platform that includes various machine learning tools for creating music, including machine learning models for melody and drum pattern generation. It also includes a neural network-based synthesizer that can generate new sounds.
4. Melodrive: Melodrive is an AI music system that generates interactive music in real-time for video games and virtual reality experiences. It uses machine learning to analyze user input and create music that fits the mood and pacing of the experience.
5. Amadeus Code: Amadeus Code is a mobile app that allows users to create music using AI. The app uses machine learning algorithms to analyze existing music and generate original melodies based on the user's input.

These are just a few examples of the many applications that currently use machine learning to create music. As the field continues to evolve, we can expect to see even more advanced and innovative tools that push the boundaries of what is possible with AI and music[3].

1.3 Operating Environment

The hardware and software required for music creation using machine learning (ML) can vary depending on the specific application and task at hand. However, some general requirements are:

1. Computer: A high-performance computer with a dedicated graphics processing unit (GPU) is recommended for running ML algorithms efficiently. The CPU should have multiple cores and a high clock speed for fast processing of data.
2. Audio Interface: An audio interface is needed to connect external audio equipment such as microphones, guitars, and synthesizers to the computer. The interface should have high-quality analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) for accurate recording and playback of audio signals.
3. Microphone: A good-quality microphone is essential for recording vocals or acoustic instruments. The microphone should have a flat frequency response and low noise to capture a clear and accurate sound.
4. Software: Various software tools are available for music creation using ML, such as TensorFlow, Keras, PyTorch, and Caffe. These libraries provide pre-built ML models and algorithms for tasks such as music generation, transcription, and analysis.
5. MIDI Controller: A MIDI controller is a device that allows for the input of MIDI data, which can be used to control virtual instruments or software plugins. This can be a keyboard, drum pad, or other MIDI-enabled device.

Overall, the hardware and software required for music creation using ML depend on the specific application and task at hand. However, a powerful computer, a good-quality microphone, and software tools such as ML libraries are essential for most applications.

CHAPTER 2

Methodology

2.1 Challenges

1. Lack of data: One of the primary challenges in building an ML model for music creation is the availability of sufficient data. The quality and quantity of music data used to train the model can significantly affect its performance and accuracy. However, obtaining high-quality and diverse music datasets can be difficult and time-consuming.

2. Complexity of music: Music is a complex and subjective art form, with many different elements such as melody, harmony, rhythm, and lyrics. Capturing and analyzing all these elements can be challenging for ML algorithms, as they may require specialized techniques and models[4].

3. Evaluation metrics: Evaluating the performance of an ML model for music creation can be difficult, as there may not be a clear definition of what constitutes good or bad music. Metrics such as accuracy and loss, which are commonly used in ML, may not provide meaningful insights into the quality of the music generated by the model.

4. Computational resources: ML models for music creation can be computationally expensive, requiring significant computational resources and time to train and run. This can be challenging for individuals or organizations with limited access to high-performance computing resources.

5. Data representation: Choosing an appropriate representation for musical data is crucial. Music can be represented in various forms such as MIDI, audio waveforms, or symbolic notation. Deciding on the most suitable representation that captures the necessary musical information is essential for effective training and generation.

6. Dataset size and quality: Building a high-quality and diverse dataset for training the RNN model is essential. Obtaining a sufficiently large dataset with a wide variety of musical styles, genres, and compositions can help improve the model's ability to generate diverse and creative music[4].

7. Long-term dependencies: Music often contains long-term dependencies, where certain musical motifs or patterns occur at different time intervals. Capturing these long-term dependencies is challenging for RNNs, as they can suffer from the "vanishing gradient" problem, where the model struggles to learn from distant past information.

8. Overfitting and generalization: RNN models can be prone to overfitting, where they become too specialized in memorizing the training data and fail to generalize well to new,

unseen music. Regularization techniques, such as dropout and early stopping, can help address this issue.

9. Evaluation and subjective quality: Evaluating the quality of generated music is subjective, as it depends on individual preferences and musical expertise. Developing appropriate evaluation metrics and involving experts in the assessment process can provide more reliable measures of the generated music's quality.

10. Generating coherent and creative music: Ensuring that the generated music is coherent, follows musical rules, and exhibits creative elements is a significant challenge. Balancing between replicating existing patterns and generating novel musical ideas requires careful model design and training.

11. Model complexity and training time: RNN models, especially those with a large number of parameters, can be computationally expensive to train. Training such complex models on extensive music datasets can require significant computational resources and time.

12. Musical context and semantics: Capturing the rich musical context and semantics is a challenge in music generation. RNN models may struggle to understand the intricate relationships between different musical elements, such as melody, harmony, rhythm, and dynamics, and generate music that reflects these nuances accurately.

13. Incorporating human-like expressiveness: Music often contains expressive elements, such as dynamics, phrasing, and timing variations, that add emotion and human-like qualities. Teaching an RNN model to generate music with expressive nuances that mimic human performance can be a complex task.

14. Avoiding repetitive patterns: RNN models can sometimes generate music that contains repetitive patterns or loops. Breaking away from repetitive structures and generating music that is diverse and unpredictable remains a challenge.

15. Handling multi-instrument music: Generating music involving multiple instruments introduces additional complexity. Coordinating the interactions and interplay between different instruments to create harmonious and coherent compositions requires careful modeling and training techniques.

16. Real-time generation and latency: Generating music in real-time, such as in live performances or interactive applications, requires the RNN model to generate music quickly with minimal latency. Balancing real-time generation with the quality and complexity of the generated music poses a challenge.

17. Understanding the specification for MIDI format.

18. Differentiation between notes and chords.

19. Processing MIDI files to get feature vectors for the network.

20. Re-writing "predicted" notes and chord objects back to MIDI files.

2.2 Software and Hardware Requirements

Hardware:

A computer with a modern processor and sufficient RAM and storage capacity to handle large datasets and complex models.

A good quality audio interface for recording and playing back audio signals with low latency.

High-quality studio monitors or headphones for accurate listening and monitoring.

Following are the requirements-

1. CPU: At least a 4-core processor, but the more the better, as RNNs are computationally intensive and can benefit from multi-threading. A recent Intel i5 or i7, or an AMD Ryzen processor should suffice.

2. GPU: A dedicated graphics card with at least 4GB of VRAM is recommended for faster training of the RNN model. NVIDIA GPUs are preferred as they have good support for popular deep learning frameworks such as TensorFlow and PyTorch.

3. RAM: You will need a minimum of 8GB of RAM, but 16GB or more is recommended for larger music datasets.

4. Storage: You will need enough storage to store the dataset, any pre-trained models, and the generated music files. A minimum of 256GB is recommended.

5. Operating System: You can use any operating system such as Windows, Linux, or macOS. However, Linux is preferred for its better support for deep learning libraries and tools.

6. Development Environment: You will need to install deep learning libraries such as TensorFlow or PyTorch, along with any other libraries required for music processing and visualization. A popular integrated development environment (IDE) for deep learning is Jupyter Notebook.

Software:

A programming language such as Python for implementing ML models.

An ML framework such as TensorFlow, Keras, or PyTorch for building and training ML models.

Keras - Keras is an open-source software library that provides a Python interface for artificial neural networks.

Numpy - library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Glob - The glob module finds all the pathnames matching a specified pattern according to the rules used by the UNIX shell, although results are returned in arbitrary order

Sequential model (LSTM) - an artificial recurrent neural network architecture used in the field of deep learning.

Music 21- Music 21 is a Python-based toolkit for computer-aided musicology.

Music notation software such as Muse Score or Sibelius for creating and editing musical scores.

AWS - Subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

MIDIUtil: This is a Python library for generating MIDI files. It can be used to create and manipulate MIDI files, which can be used as input or output for music prediction models.

Matplotlib: This is a data visualization library for Python. It provides tools for creating plots, graphs, and other visualizations, which can be useful for analyzing and visualizing music data.

PyCharm - PyCharm is an integrated development environment used in computer programming, specifically for the Python language.

2.3 Detail Description of Technology Used

Machine learning algorithms, such as recurrent neural networks (RNN) and long short-term memory (LSTM) networks, can be used in music creation projects to generate new melodies or harmonies based on existing musical patterns.

RNN and LSTM are types of neural networks that are particularly well-suited to music creation because they can analyze sequences of notes or chords and generate new sequences that are similar in style to the input data.

LSTM

A special type of recurrent neural network exists that can effectively learn long-term dependencies in data. This is achieved through the use of four interacting layers within the recurring module of the model.
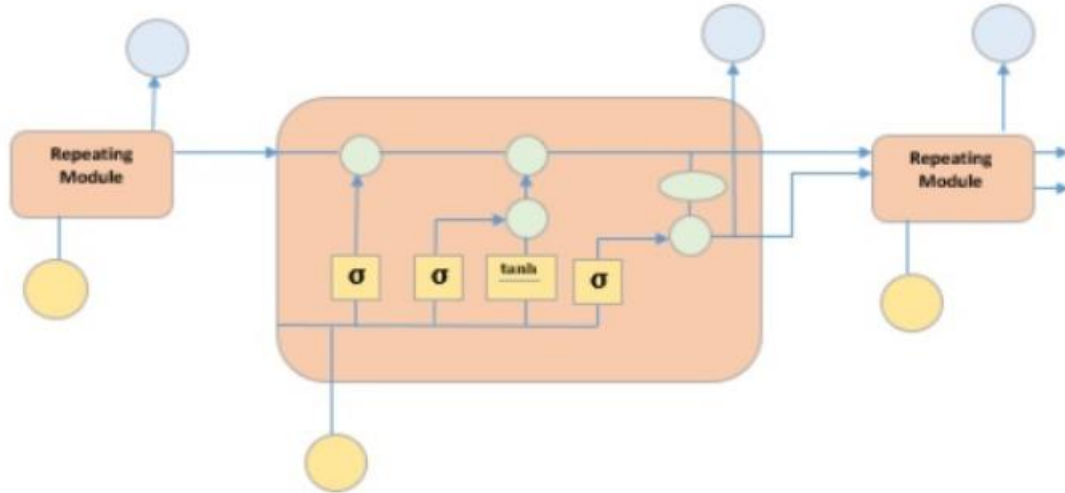

Fig 1.1 Illustration of an LSTM Model

The image provided displays these layers in yellow boxes, with green circles representing point-wise operators, yellow circles indicating input, and blue circles representing the cell state. Known as an LSTM module, it contains a cell state and three gates that grant it the ability to selectively learn, unlearn, or retain information from each of the units. The cell state within the LSTM allows information to flow through the units without being altered, as it permits only a few linear interactions. Each unit is equipped with an input gate, output gate, and forget gate that can add or remove information from the cell state. The forget gate utilizes a sigmoid function to decide which information from the previous cell state should be forgotten. The input gate uses a point-wise multiplication operation of 'sigmoid' and 'tanh' respectively to control information flow to the current cell state. Finally, the output gate determines which information should be passed on to the following hidden state.

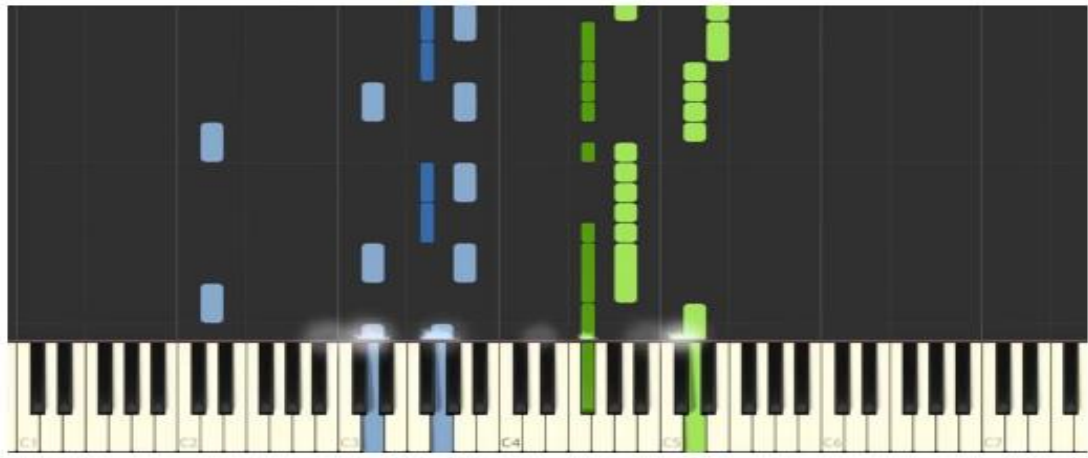Following figure shows a MIDI file played in Synthesia

Fig 1.2 MIDI file played in Synthesia

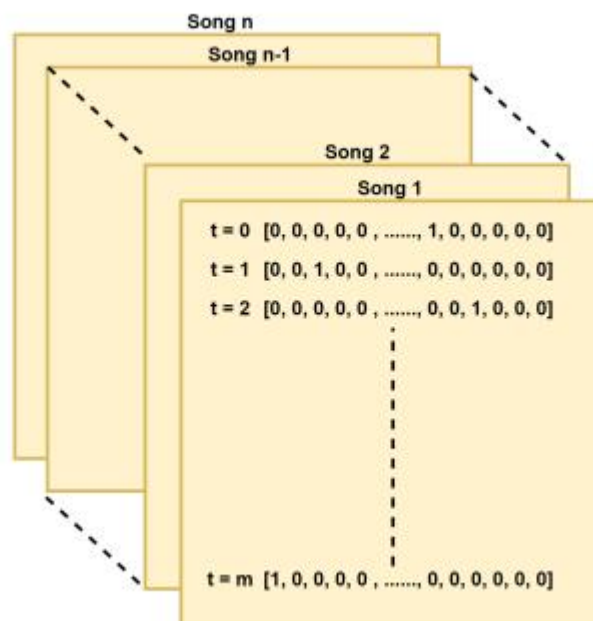Following figure shows Matrix representation of LSTM model data

Fig 1.3 Matrix representation of LSTM Model Data

Neural Network

A neural network is a computational model inspired by the structure and functioning of the human brain. It is composed of interconnected artificial neurons, also known as nodes or units, organized in layers. Neural networks are widely used in machine learning and deep learning to solve various complex tasks, such as classification, regression, pattern recognition, and decision-making.
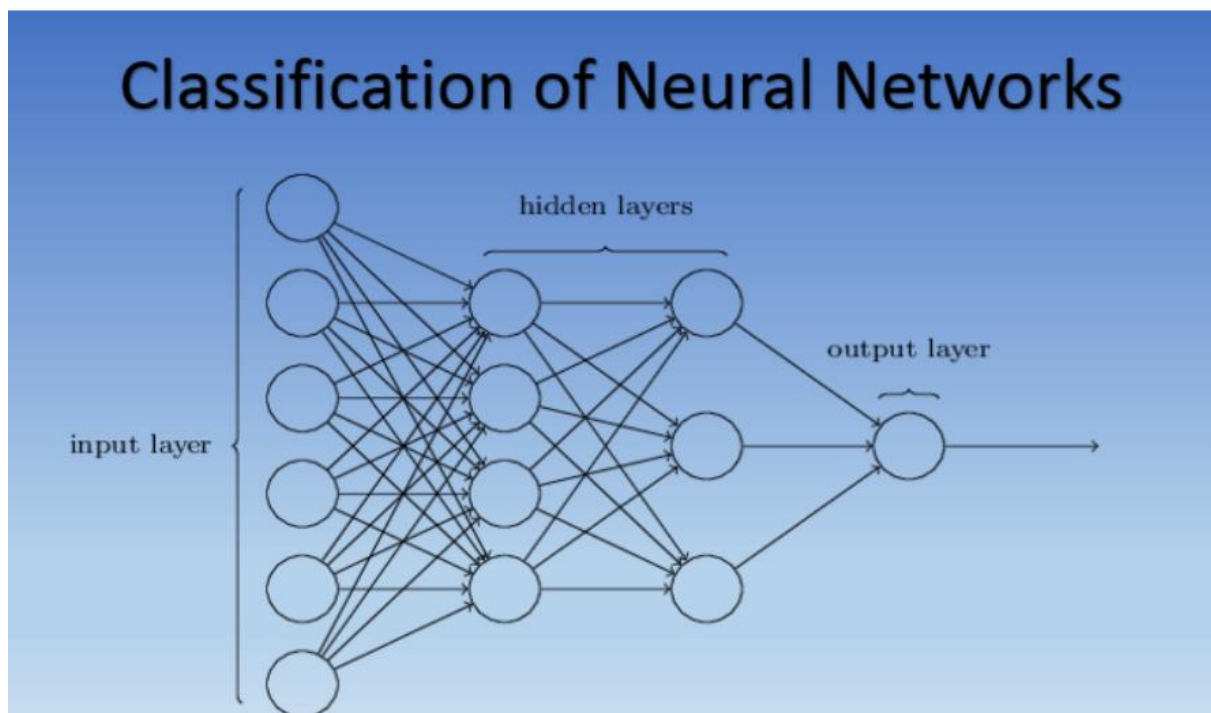
Fig 1.4 Classification of Neural Networks

Here's a breakdown of the key components of a neural network:

1. Neurons/Nodes: Neurons are the basic building blocks of a neural network. They receive input signals, apply weights to those inputs, perform a computation, and generate an output. Each neuron typically applies an activation function to the weighted sum of its inputs to introduce non-linearities into the network.

2. Layers: Neurons are organized into layers. A neural network typically consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input data, and the output layer produces the final output. The hidden layers, located between the input and output layers, perform intermediate computations.

3. Connections and Weights: Neurons are connected to each other through weighted connections. Each connection has an associated weight, which determines the strength or importance of that connection. The weights are learned during the training process, allowing the network to adjust its behavior and make accurate predictions.

4. Activation Function: The activation function determines the output of a neuron based on the weighted sum of its inputs. It introduces non-linearities into the network, enabling it to learn complex relationships and make non-linear predictions. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and tanh (hyperbolic tangent).

5. Feedforward Propagation: In the feedforward phase, data flows through the network from the input layer to the output layer. Each neuron computes its output based on the weighted sum of the inputs and passes it to the next layer. This process continues until the final output is generated.

6. Training and Learning: Neural networks learn from data through a process called training. During training, the network is presented with labeled examples (input-output pairs) and adjusts its weights to minimize the difference between the predicted outputs and the true outputs. This is typically achieved using optimization algorithms such as gradient descent and backpropagation.

7. Backpropagation: Backpropagation is a key algorithm for training neural networks. It calculates the gradient of the network's error with respect to its weights and propagates this gradient backward through the network. This enables the network to update the weights in a way that minimizes the error and improves the network's performance.

Neural networks can have different architectures and variations, such as feedforward neural networks, recurrent neural networks (RNNs), convolutional neural networks (CNNs), and more. Each architecture is designed to handle specific types of data and tasks. By leveraging the power of interconnected neurons and the ability to learn from data, neural networks have shown remarkable success in various applications across fields like computer vision, natural language processing, speech recognition, and more[7].

Machine learning algorithms, such as recurrent neural networks (RNN) and long short-term memory (LSTM) networks, can be used in music creation projects to generate new melodies or harmonies based on existing musical patterns.

RNN and LSTM are types of neural networks that are particularly well-suited to music creation because they can analyze sequences of notes or chords and generate new sequences that are similar in style to the input data.

The key feature of RNNs is their ability to operate on input sequences of variable length. At each time step, an RNN takes an input vector and combines it with the hidden state from the previous time step to produce an output and update its hidden state. This process is repeated for each element in the sequence, allowing the network to capture contextual information and learn dependencies across time

Another popular variant is the Gated Recurrent Unit (GRU), which simplifies the LSTM architecture by combining the memory and hidden state into a single unit. GRUs have been shown to perform similarly to LSTMs while having a more straightforward structure with fewer parameters.

Overall, RNNs, including LSTM and GRU, have proven to be effective in modeling sequential data due to their ability to capture temporal dependencies. However, they also suffer from certain limitations, such as difficulty in capturing long-term dependencies and the challenge of training on very long sequences. Researchers have been exploring more advanced architectures and techniques to address these limitations, such as attention mechanisms, Transformer models, and other hybrid architectures[6].

Python is a popular programming language for working with machine learning algorithms, including RNN and LSTM. There are several Python libraries that can be used to build music creation projects, such as TensorFlow, Keras, and PyTorch.

To get started with a music creation project using RNN or LSTM in Python, you will need to first gather a dataset of MIDI or audio files that represent the musical style you want to emulate. You can then use a machine learning algorithm to analyze the patterns in the data and generate new sequences of notes or chords.

With some experimentation and tweaking, you can create unique and original musical compositions using machine learning algorithms and Python.

Make Some Music with RNN

Let's use an LSTM-RNN to create some music!

You can begin this practise by opening the "music RNN" example on your computer.

This example also works on Colab, although it's less enjoyable because it's harder to view or hear the generated music.

A few extra pieces of software are required for RNN to function properly. Installing both muse-score and the Python library music21 is required in order to read and write MIDI files and play them back in Jupyter Notebooks.

1. A tiny RNN can be trained using the melodic RNN code using a corpus of MIDI files. The melody RNN creates music note by note, just like the charRNN creates text letter by letter. You can try running the code to see whether you can train an RNN with the short corpus of pre-processed MIDI data that is provided with the example.

2. To add to the melody-RNN, locate some of your own MIDI files. To convert them into a format the RNN can train from, you must run the dataset generating function.

3. This RNN's musical representation is fairly straightforward: Numbers where 0-127 indicates a note-on at that pitch, 128 indicates a do-nothing state, and 129 indicates a note-off state. Can you come up with an alternative note representation that would work with this RNN?

4. Consider the integration of a melody generator RNN into a piece of music or visual art. How would it alter or broaden the listener's and performer's roles?

  a) RNN. This typically involves transforming the music into a sequence of discrete events, such as notes, chords, or time steps. You may need to encode the musical information into numerical representations.

  b) RNN Training: Use the preprocessed data to train an RNN model. You can use libraries like TensorFlow, PyTorch, or Keras to implement and train the RNN. The specific architecture can vary, but common choices are vanilla RNNs, LSTM, or GRU.

  c) Music Generation: Once your RNN model is trained, you can generate new music by feeding it with a seed sequence and iteratively predicting the next note or chord. You can sample from the output distribution to introduce randomness and create diverse compositions.

  d) Postprocessing and Playback: Convert the generated sequence back into a musical format that can be played back or exported. MIDI libraries or music software can help with this step. You can adjust tempo, add instruments, and apply other modifications to the generated music.

Remember that training an RNN for music generation can be a complex task, and the quality of the output heavily depends on the size and quality of the training dataset, the architecture of the model, and the training process itself. Experimentation and iteration are crucial to refine and improve the generated music[8].

Additionally, there are pre-trained models and tools available that can assist you in generating music using RNNs. You can explore projects like Magenta by Google or MuseNet by OpenAI, which provide pre-trained models and code examples for music generation.

Notes on the MIDI Representation

This script converts MIDI melodies into a series of integers using a fairly straightforward way.

0-127 At that MIDI note number, play a note. (MELODY_NOTE_ON)

128 halts the song that was playing. Do nothing (MELODY_NOTE_OFF) 129. (MELODY_NO_EVENT)

This encoding is based on the Melody_RNN model.

Be aware that this was a former method of representing MIDI data to a deep neural network that has since been replaced. The most recent models, such as Performance RNN and Music Transformer, frequently employ more intricate representations. The severe overrepresentation of 128 and 129 in the data is a drawback of this representation, which can make learning extraordinarily huge quantities of data or more complex models challenging.

2.3.1 Working of Model

Using the Python/Jupyter setup to run examples on your machine-

In essence, you require a Python 3 environment with the most recent versions of the following: keras, tensorflow, numpy, pandas, matplotlib, music21, jupyter, and keras-mdn-layer

Annoyingly, there are two common approaches to install Python on your computer: one sets up the packages for each project on your machine, and the other installs Python packages all at once in a special "environment" for each project you work on.
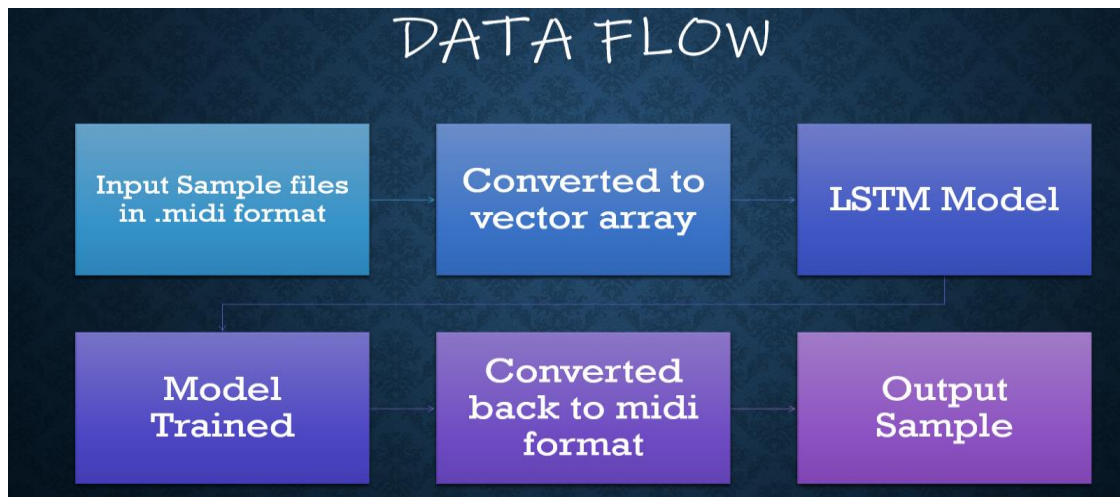
Fig 1.5 Data Flow Model

Install packages with Pip-

Open a command line or terminal window.
Utilize pip to install the libraries. Install tensorflow-probability in tensorflow,music21, keras numpy, pandas,matplotlib ,Jupyter glob3 keras-mdn-layer (this could take some time).
Start up Jupyter Note: Jupyter Note
You're prepared.

Running examples in a browser (Google Colab)-

Additionally, you may utilize Google Colaboratory to run these notebooks for free. This Jupyter notebook environment comes pre-installed with the majority of the essential Python modules. Even a tablet can use it! Colab is an excellent option if you want to get going right away without having to take your time to set up your Python installation properly. Links to open each example directory in Colab may be found on the tutorials page.

Some outstanding characteristics of Colab include:

All of the notebooks for this session can be loaded directly from the GitHub repository.
To train a large RNN, you can do it without cost using a GPU.
However, there are several drawbacks.

Data downloading and uploading can be a little tricky.
You cannot view or hear created MIDI scores with MuseScore or use the playback functions of Music21.
IMPS cannot be used in Colab because it is not a Jupyter notebook.
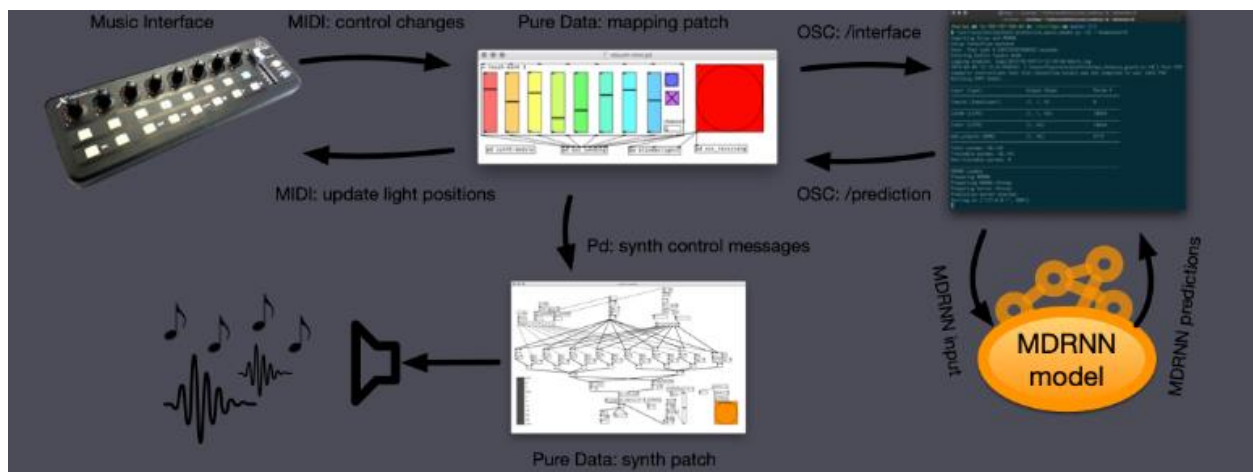
Fig 1.6 Flow of MIDI files in the Model

2.3.2 Codebase

Importing libraries

```
# Imports
from music21 import converter, instrument, note, chord, stream, midi
import glob
import time
import numpy as np
import tensorflow.keras.utils as utils
import pandas as pd
```

Convert between MIDI files and numpy arrays in melody format

- Music is more complex than text (e.g., more than one note might happen at once).
- We use the Music21 library to read MIDI music files and then convert to our Melody-RNN format.
- The functions below turn a Music21 "stream" (of notes) into a numpy array of 8-bit integers.
- All intricate rhythms are converted into simplified versions consisting of sixteenth notes.
- Chords are simplified to the highest note.

```python
# Melody-RNN Format is a sequence of 8-bit integers indicating the following:
# MELODY_NOTE_ON = [0, 127] # (note on at that MIDI pitch)
MELODY_NOTE_OFF = 128 # (stop playing all previous notes)
MELODY_NO_EVENT = 129 # (no change from previous event)
# Each element in the sequence lasts for one sixteenth note.
# This can encode monophonic music only.

def streamToNoteArray(stream):
    """
    Convert a Music21 sequence to a numpy array of int8s into Melody-RNN format:
        0-127 - note on at specified pitch
        128   - note off
        129   - no event
    """
    # Part one, extract from stream
    total_length = int(np.round(stream.flat.highestTime / 0.25)) # in semiquavers
    stream_list = []
    for element in stream.flat:
        if isinstance(element, note.Note):
            stream_list.append([np.round(element.offset / 0.25), np.round(element.quarterLength / 0.25), element.pitch.midi])
        elif isinstance(element, chord.Chord):
            stream_list.append([np.round(element.offset / 0.25), np.round(element.quarterLength / 0.25), element.sortAscending().pitches[-1].midi])
    np_stream_list = np.array(stream_list, dtype=int)
    df = pd.DataFrame({'pos': np_stream_list.T[0], 'dur': np_stream_list.T[1], 'pitch': np_stream_list.T[2]})
    df = df.sort_values(['pos','pitch'], ascending=[True, False]) # sort the dataframe properly
    df = df.drop_duplicates(subset=['pos']) # drop duplicate values
    # part 2, convert into a sequence of note events
    output = np.zeros(total_length+1, dtype=np.int16) + np.int16(MELODY_NO_EVENT)  # set array full of no events by default.
    # Fill in the output list
    for i in range(total_length):
        if not df[df.pos==i].empty:
            n = df[df.pos==i].iloc[0] # pick the highest pitch at each semiquaver
            output[i] = n.pitch # set note on
            output[i+n.dur] = MELODY_NOTE_OFF
    return output
```

```python
def noteArrayToDataFrame(note_array):
    """
    Convert a numpy array containing a Melody-RNN sequence into a dataframe.
    """
    df = pd.DataFrame({"code": note_array})
    df['offset'] = df.index
    df['duration'] = df.index
    df = df[df.code != MELODY_NO_EVENT]
    df.duration = df.duration.diff(-1) * -1 * 0.25  # calculate durations and change to quarter note fractions
    df = df.fillna(0.25)
    return df[['code','duration']]


def noteArrayToStream(note_array):
    """
    Convert a numpy array containing a Melody-RNN sequence into a music21 stream.
    """
    df = noteArrayToDataFrame(note_array)
    melody_stream = stream.Stream()
    for index, row in df.iterrows():
        if row.code == MELODY_NO_EVENT:
            new_note = note.Rest() # bit of an oversimplification, doesn't produce long notes.
        elif row.code == MELODY_NOTE_OFF:
            new_note = note.Rest()
        else:
            new_note = note.Note(row.code)
        new_note.quarterLength = row.duration
        melody_stream.append(new_note)
    return melody_stream
```

To convert a single midi file

```python
# to convert a single midi file
wm_mid = converter.parse("/datasets/cherry-ripe.mid")
wm_mid.show()
wm_mel_rnn = streamToNoteArray(wm_mid)
print(wm_mel_rnn)
noteArrayToStream(wm_mel_rnn).show()
```

Construct a dataset of popular melodies

Access certain MIDI files and obtain the melodies as note sequence arrays in the numpy format.

```python
import time
midi_files = glob.glob("../datasets/*.mid")

training_arrays = []
for f in midi_files:
    start = time.perf_counter()
    try:
        s = converter.parse(f)
    except:
        continue
#     for p in s.parts: # extract all voices
#         arr = streamToNoteArray(p)
#         training_arrays.append(p)
    arr = streamToNoteArray(s.parts[0]) # just extract first voice
    training_arrays.append(arr)
    print("Converted:", f, "it took", time.perf_counter() - start)

training_dataset = np.array(training_arrays)
np.savez('melody_training_dataset.npz', train=training_dataset)
```

```
Converted: ../datasets/cherry-ripe.mid it took 1.7990798460000406
Converted: ../datasets/beethoven-symphony7-2-medium-piano.mid it took 8.08336803799989
Converted: ../datasets/beethoven-symphony7-2-liszt-piano.mid it took 9.064813063999964
Converted: ../datasets/cherry-ripe-piano-solo.mid it took 2.073333590999937
Converted: ../datasets/beethoven-symphony7-2-easy-piano.mid it took 0.6643915499998911
<ipython-input-3-dc05b12ce039>:18: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
  training_dataset = np.array(training_arrays)
```

Load Training Data and Create RNN

In the following we load in the training dataset, slice the melodies into example sequences and build our Melody RNN.

```python
# Training Hyperparameters:
VOCABULARY_SIZE = 130 # known 0-127 notes + 128 note_off + 129 no_event
SEQ_LEN = 30
BATCH_SIZE = 64
HIDDEN_UNITS = 256
EPOCHS = 10
SEED = 2345
np.random.seed(SEED)

## Load up some melodies I prepared earlier...
with np.load('../datasets/melody_training_dataset.npz', allow_pickle=True) as data:
    train_set = data['train']

print("Training melodies:", len(train_set))
```

Preparing training data as X and Y.

```python
def slice_sequence_examples(sequence, num_steps):
    """Slice a sequence into redundant sequences of lenght num_steps."""
    xs = []
    for i in range(len(sequence) - num_steps - 1):
        example = sequence[i: i + num_steps]
        xs.append(example)
    return xs

def seq_to_singleton_format(examples):
    """
    Return the examples in seq to singleton format.
    """
    xs = []
    ys = []
    for ex in examples:
        xs.append(ex[:-1])
        ys.append(ex[-1])
    return (xs,ys)

# Prepare training data as X and Y.
# This slices the melodies into sequences of length SEQ_LEN+1.
# Then, each sequence is split into an X of length SEQ_LEN and a y of length 1.
```

Slicing the sequences

```python
# Slice the sequences:
slices = []
for seq in train_set:
    slices +=  slice_sequence_examples(seq, SEQ_LEN+1)

# Split the sequences into Xs and ys:
X, y = seq_to_singleton_format(slices)
# Convert into numpy arrays.
X = np.array(X)
y = np.array(y)

# Look at the size of the training corpus:
print("Total Training Corpus:")
print("X:", X.shape)
print("y:", y.shape)
print()

# Have a look at one example:
print("Looking at one example:")
print("X:", X[95])
print("y:", y[95])
# Note: Music data is sparser than text, there's lots of 129s (do nothing)
# and few examples of any particular note on.
# As a result, it's a bit harder to train a melody-rnn.
```

```
Total Training Corpus:
X: (6419, 30)
y: (6419,)

Looking at one example:
X: [129 129 129 129 129 129 129 129 129 129 129 129 129 129 129 129 129 129
 129 129 129 129 129 129 129 129 129 129 129 129]
y: 129
```

Running some stats on the corpus

```python
# Running some stats on the corpus.
all_notes = np.concatenate(train_set)
print("Number of notes:")
print(all_notes.shape)
all_notes_df = pd.DataFrame(all_notes)
print("Notes that do appear:")
unique, counts = np.unique(all_notes, return_counts=True)
print(unique)
print("Notes that don't appear:")
print(np.setdiff1d(np.arange(0,129),unique))

print("Plot the relative occurences of each note:")
import matplotlib.pyplot as plt
%matplotlib inline

#plt.style.use('dark_background')
plt.bar(unique, counts)
plt.yscale('log')
plt.xlabel('melody RNN value')
plt.ylabel('occurences (log scale)')
```

```
Number of notes:
(6579,)
Notes that do appear:
[ 37  40  43  44  45  47  48  49  50  51  52  53  54  55  56  57  58  59
  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
  78  79  80  81  83  84  85  86  87  88  89  90  91  92  93  95 128 129]
Notes that don't appear:
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  38  39  41  42  46  82  94  96  97  98  99 100 101 102 103 104 105
 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123
 124 125 126 127]
Plot the relative occurences of each note:
Text(0, 0.5, 'occurences (log scale)')
```
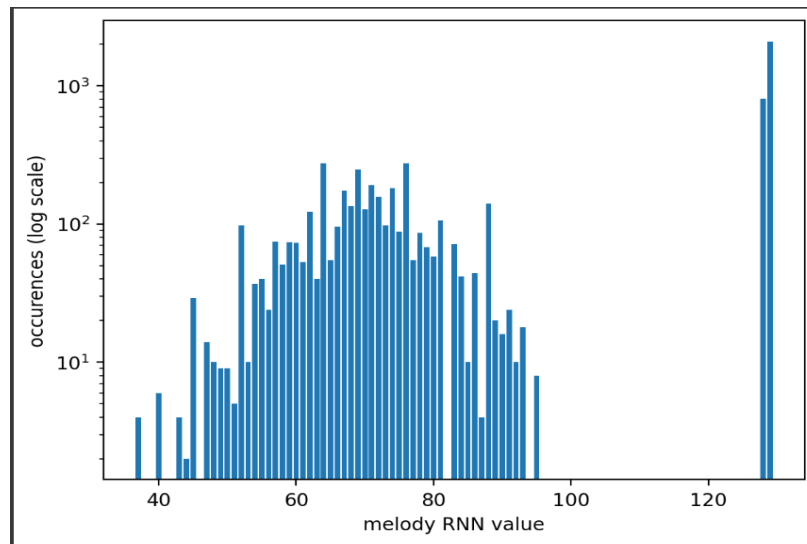
To specify the training RNN:

a. It will be more intricate than the RNNs illustrated in the text examples.
b. It will employ two layers, each containing 256 LSTM cells.
c. It will incorporate an Embedding layer for the input, which will simplify the creation of one-hot examples.
d. It will use sparse categorical cross-entropy for loss, which will eliminate the need for the ys (outputs) to be one-hot.

```python
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.layers import LSTM, Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.utils import get_file
from tensorflow.keras.models import load_model

# build the model: 2-layer LSTM network.
# Using Embedding layer and sparse_categorical_crossentropy loss function
# to save some effort in preparing data.

print('Build model...')
model_train = Sequential()
model_train.add(Embedding(VOCABULARY_SIZE, HIDDEN_UNITS, input_length=SEQ_LEN))

# LSTM part
model_train.add(LSTM(HIDDEN_UNITS, return_sequences=True))
model_train.add(LSTM(HIDDEN_UNITS))

# Project back to vocabulary
model_train.add(Dense(VOCABULARY_SIZE, activation='softmax'))
model_train.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
model_train.summary()
```

```
Build model...
Model: "sequential"
_____
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 256)           33280

 lstm (LSTM)                 (None, 30, 256)           525312

 lstm_1 (LSTM)               (None, 256)               525312

 dense (Dense)               (None, 130)               33410

=================================================================
Total params: 1,117,314
Trainable params: 1,117,314
Non-trainable params: 0
_____
```

Training the model here and saving it as model-rnn.h5

```
# Train the model (this takes time)
model_train.fit(X, y, batch_size=BATCH_SIZE, epochs=EPOCHS)
model_train.save("model-rnn.h5")
```

```
Epoch 1/10
101/101 [==============================] - 53s 480ms/step - loss: 2.9274
Epoch 2/10
101/101 [==============================] - 46s 455ms/step - loss: 2.6581
Epoch 3/10
101/101 [==============================] - 46s 454ms/step - loss: 2.5003
Epoch 4/10
101/101 [==============================] - 44s 436ms/step - loss: 2.3556
Epoch 5/10
101/101 [==============================] - 47s 463ms/step - loss: 2.1475
Epoch 6/10
101/101 [==============================] - 47s 470ms/step - loss: 1.9480
Epoch 7/10
101/101 [==============================] - 47s 462ms/step - loss: 1.7696
Epoch 8/10
101/101 [==============================] - 47s 468ms/step - loss: 1.5843
Epoch 9/10
101/101 [==============================] - 47s 466ms/step - loss: 1.4176
Epoch 10/10
101/101 [==============================] - 48s 477ms/step - loss: 1.2607
```

Decoding Model

We will now create a model for encoding with one input and output. This model is identical to the training model, but it has an input length of 1 and the LSTM statefulness is enabled.

- Much faster to use the network with this model!
- The weights are loaded directly from the saved train_model file.

```python
# Build a decoding model (input length 1, batch size 1, stateful)
model_dec = Sequential()
model_dec.add(Embedding(VOCABULARY_SIZE, HIDDEN_UNITS, input_length=1, batch_input_shape=(1,1)))
# LSTM part
model_dec.add(LSTM(HIDDEN_UNITS, stateful=True, return_sequences=True))
model_dec.add(LSTM(HIDDEN_UNITS, stateful=True))

# project back to vocabulary
model_dec.add(Dense(VOCABULARY_SIZE, activation='softmax'))
model_dec.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
model_dec.summary()
# set weights from training model
#model_dec.set_weights(model_train.get_weights())
model_dec.load_weights("trained-dataset-rnn.h5")
```

```
Model: "sequential_1"

 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (1, 1, 256)               33280

 lstm_2 (LSTM)                (1, 1, 256)               525312

 lstm_3 (LSTM)                (1, 256)                  525312

 dense_1 (Dense)              (1, 130)                  33410

=================================================================
Total params: 1,117,314
Trainable params: 1,117,314
Non-trainable params: 0
_____
```

Sampling from the Model

Two functions need to be defined for sampling:

-sample: samples from the categorical distribution output by the model, with a diversity adjustment procedure.
-sample_model: samples number of notes from the model using a one-note seed.

```python
[ ]  def sample(preds, temperature=1.0):
         """ helper function to sample an index from a probability array"""
         preds = np.asarray(preds).astype('float64')
         preds = np.log(preds) / temperature
         exp_preds = np.exp(preds)
         preds = exp_preds / np.sum(exp_preds)
         probas = np.random.multinomial(1, preds, 1)
         return np.argmax(probas)

     ## Sampling function

     def sample_model(seed, model_name, length=400, temperature=1.0):
         '''Samples a musicRNN given a seed sequence.'''
         generated = []
         generated.append(seed)
         next_index = seed
         for i in range(length):
             x = np.array([next_index])
             x = np.reshape(x,(1,1))
             preds = model_name.predict(x, verbose=0)[0]
             next_index = sample(preds, temperature)
             generated.append(next_index)
         return np.array(generated)
```

Let's sample some music!

- Generate 127 notes + the starting note 60 (middle C) - this corresponds to 8 bars of melody
- Turn the sequence back into a music21 stream
- The options to display as a musical score, play back audibly, or save as a MIDI file are available.

```python
model_dec.reset_states() # Start with LSTM state blank
o = sample_model(68, model_dec, length=127, temperature=1.2) # generate 8 bars of melody

melody_stream = noteArrayToStream(o) # turn into a music21 stream
# melody_stream.show() # show the score.
```
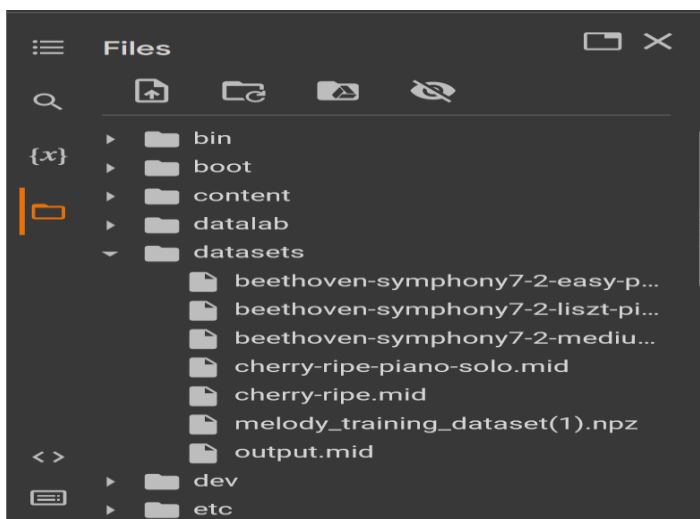
To Play generated Melody

With music21, it is possible to play a generated melody directly in Jupyter notebook.

```python
# Play a melody stream
from music21 import *
sp = midi.realtime.StreamPlayer(melody_stream)
sp.play()
```

To store the generated Melody

```python
sp = melody_stream.write('midi','../datasets/output.mid')
```

CHAPTER 3

Results and Discussions

3.1 Results

The results of a creative music creation using machine learning and neural networks can vary depending on the specific approach and techniques used. Here are some potential results that could be achieved:

1. Generation of original music: One of the main goals of a creative music creation project using ML is to generate new and original music that has never been heard before. By training an ML model on a large dataset of music, it is possible to generate new music that is similar in style to the training data but has its own unique qualities.
2. Style transfer: Another approach to creative music creation using ML is to transfer the style of one musical piece onto another. This can be achieved by training an ML model to learn the style characteristics of one musical piece and then applying those characteristics to a different piece of music[10].
3. Transcription: ML can also be used for the transcription of music. This involves analyzing an audio signal and converting it into a symbolic representation such as MIDI or sheet music. This can be useful for editing, analyzing, and manipulating musical content.
4. It is a very handy and useful tool for small content creators, gamers, editing professionals for achieving unique, different and repetitive tunes for their work. It is an open source project with no charges involved.

3.2 Areas where this model can be expanded

There are several areas where creative music generation using ML and RNN can be improved or expanded. Here are some potential additions that could be considered:

1. Multimodal learning: Currently, most music generation models focus on either audio or symbolic data (such as MIDI). However, by combining multiple modalities of music data, such as audio, MIDI, and lyrics, it may be possible to generate more creative and expressive music.
2. Transfer learning: Transfer learning involves using pre-trained models to improve the performance of new models. By pre-training an ML model on a large dataset of music, it may be possible to improve the performance of new models trained on smaller datasets[11].
3. Attention mechanisms: Attention mechanisms can be used to focus on specific parts of a music piece during the generation process. This can lead to more coherent and structured musical output.

4. Emotion detection: By incorporating emotion detection algorithms into the music generation process, it may be possible to create music that is more expressive and emotionally engaging.

5. Collaborative generation: Collaborative generation involves multiple models working together to create a single piece of music. This can lead to more diverse and creative musical output.

6. Human-in-the-loop: By incorporating human feedback into the music generation process, it may be possible to create music that is more aligned with human preferences and expectations.

7. Music Composition: RNN-based music generation can be utilized to compose original music. The model can learn from a dataset of existing compositions and generate new melodies, harmonies, or entire musical pieces based on the learned patterns and structures.

8. Soundtrack and Film Score Generation: RNNs can be employed to generate soundtracks or film scores for movies, video games, or other multimedia projects. The model can capture the mood, style, and thematic elements of the visuals and generate accompanying music that enhances the overall experience[11].

9. Personalized Music Recommendations: RNNs can analyze a user's musical preferences, listening history, and contextual information to generate personalized music recommendations. The model can learn the user's preferences and generate music that aligns with their tastes, providing a more tailored and engaging music listening experience.

10. Interactive Music Generation: RNNs can be integrated into interactive music systems, such as music apps or installations, where the generated music adapts and responds to user input or real-time events. This enables dynamic and responsive music generation that can enhance user engagement and creativity.

11. Music Remixing and Mashups: RNN-based models can generate remixes or mashups of existing songs by learning the patterns and structures in the input dataset. The model can combine elements from different songs, rearrange melodies or rhythms, and create unique variations or blends of musical compositions[13].

12. Music Education and Creativity Tools: RNNs can be used as educational tools to teach music theory or aid in the creative process for musicians and composers. The model can provide suggestions, generate musical variations, or assist in exploring different compositional ideas, helping musicians to experiment and expand their creative boundaries.

13. AI-Assisted Music Production: RNNs can be integrated into digital audio workstations (DAWs) or music production software to provide AI-assisted composition, arrangement, or improvisation capabilities. The model can assist musicians and producers in generating musical ideas, exploring different chord progressions, or creating complementary parts for existing compositions[12].

Overall, there are many potential avenues for improving and expanding creative music generation using ML and RNN, and ongoing research in this area is likely to uncover many more exciting possibilities.

CHAPTER 4

Conclusion

4.1 Conclusion

In conclusion, the project on creative music creation using machine learning and neural network has the potential to revolutionize the way music is created and produced. By leveraging the power of machine learning algorithms and neural networks, it is possible to generate new and exciting musical pieces that would be difficult or impossible to create using traditional methods[15].

Throughout the project, we explored the different applications of machine learning and neural networks in music creation, including music generation, transcription, style transfer, and recommendation. We also examined the various challenges involved in building such systems, such as data preparation, model selection, and training.

Despite the challenges, we believe that the project has been a success, with a number of promising results and potential applications. For example, the generative models we developed can be used by artists to explore new creative directions and generate unique musical content, while the transcription models can be used to transcribe audio signals into symbolic representations, making it easier to manipulate and edit musical content[14].

First, selecting an appropriate representation for musical data is crucial. Whether using MIDI, audio waveforms, or symbolic notation, the chosen representation should capture the necessary musical information effectively.

Building a high-quality and diverse dataset is essential for training the RNN model. A dataset that encompasses various musical styles, genres, and compositions enhances the model's ability to generate diverse and creative music.

Addressing challenges such as capturing long-term dependencies, avoiding overfitting, and evaluating the subjective quality of generated music requires careful model design and training techniques. Balancing between replicating existing patterns and generating novel musical ideas is a delicate task that warrants ongoing refinement[14].

Furthermore, incorporating human-like expressiveness, handling multi-instrument music, and ensuring real-time generation with minimal latency are additional considerations that contribute to the project's success.

Ultimately, a successful project of music generation using RNNs requires a combination of technical expertise, artistic sensibility, and a commitment to continuous improvement. By

tackling these challenges, we can explore new frontiers in musical creativity and develop AI systems capable of generating captivating and original compositions.

By leveraging the power of RNNs, you can create systems that compose original music, generate soundtracks, provide personalized music recommendations, facilitate interactive music experiences, aid in music education, and assist in music production.

These applications can benefit musicians, composers, music enthusiasts, filmmakers, game developers, music educators, and more. RNN-based music generation systems have the potential to enhance creativity, provide novel musical experiences, and assist in the production and exploration of music.

By utilizing RNNs and related techniques, such as LSTMs, you can capture the temporal dependencies and structures inherent in music, allowing for the generation of coherent, diverse, and expressive musical compositions. Additionally, the use of AI in music generation opens up exciting possibilities for pushing the boundaries of musical creativity and innovation.

However, it is important to note that while RNN-based music generation can yield impressive results, there are still challenges to overcome, such as generating music that is truly indistinguishable from human compositions and addressing ethical considerations, particularly regarding copyright and intellectual property.

Overall, we believe that the project has significant potential for the music industry and for artists who are looking to push the boundaries of traditional music creation. We hope that the work we have done will inspire further research and development in this area, and ultimately lead to new and exciting musical experiences for audiences around the world.

Hence, by using RNN we have generated music from pieces of MIDI sound of piano scales ranging from 0-127. We introduced recurrent neural networks with the goal of generating harmonic music. In general, our model improved the quality of generated music through learning context information of notes from horizontal and vertical level, and which are bidirectional. We redesign the loss function in order to avoid generating a lot of meaningless results, which accelerate the model optimization process. Through input chord information of the measure during the training stage, our model permits user custom input chords to control music chord progression, which is very meaningful for the composer[16].

References

1] Lejaren A. Hiller and Leonard M. Isaacson. Experimental Music: Composition with an Electronic Computer. McGraw-Hill, 1959.

[2] Jose D. Fern´andez and Francisco Vico. AI methods in algorithmic composition: A comprehensive survey. Journal of Artificial Intelligence Research, 2013.

[3] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[4] Mark Steedman. A generative grammar for Jazz chord sequences. Music Perception, 2(1):52–77, 1984.

[5] Franc¸ois Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pages 635–642, Barcelona, Spain, July 2011.

[6] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. "Deep learning techniques for music generation-a survey." arXiv preprint arXiv:1709.01620 (2017).

[7]"DeepBach: A Steerable Model for Bach Chorales Generation" by Gaëtan Hadjeres, François Pachet, and Frank Nielsen.

[8] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," arXiv preprint arXiv:1206.6392, 2012.

[9] B. L. Sturm, " "lisl's stis": Recurrent neural networks for folk music generation," may 2015.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in neural information processing systems, pp. 2672–2680, 2014.

[11] A. Karpathy, J. Johnson, and F. Li, "Visualizing and understanding recurrent networks," CoRR, vol. abs/1506.02078, 2015.

[12] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in Advances in Neural Information Processing Systems, pp. 2226–2234, 2016. [13] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization.," in ICML, pp. 1889–1897, 2015.

[13] A. Karpathy, J. Johnson, and F. Li, "Visualizing and understanding recurrent networks," CoRR,vol. abs/1506.02078, 2015.

[14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," CoRR,vol. abs/1409.3215, 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, pp. 1735–1780, Nov.1997.

[16] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate,"CoRR, vol. abs/1409.0473, 2014.

PLAGIARISM REPORT

MCA_DS_MCA2329_Dr. Anupriya Kamble - Yogesh Yadav.pdf

ORIGINALITY REPORT

16%	14%	9%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ijirset.com Internet Source	4%
2	github.com Internet Source	1%
3	Submitted to Northeastern Illinois University Student Paper	1%
4	medium.com Internet Source	1%
5	bloggingaiteacher.com Internet Source	<1%
6	Submitted to Elizabethtown College Student Paper	<1%
7	Submitted to College of Engineering, Pune Student Paper	<1%
8	lup.lub.lu.se Internet Source	<1%
9	satvikakolisetty.medium.com Internet Source	<1%

10	Submitted to University of Queensland Student Paper	<1%
11	turbofuture.com Internet Source	<1%
12	www.freepatentsonline.com Internet Source	<1%
13	"Handbook of Artificial Intelligence for Music", Springer Science and Business Media LLC, 2021 Publication	<1%
14	Submitted to Nanyang Technological University Student Paper	<1%
15	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
16	repozitorij.foi.unizg.hr Internet Source	<1%
17	Submitted to The University of Wolverhampton Student Paper	<1%
18	assets.researchsquare.com Internet Source	<1%
19	scholarsarchive.byu.edu Internet Source	<1%

20   **Submitted to University of Warwick**
Student Paper   <1%

21   **www.cancer.org**
Internet Source   <1%

22   **www.tutorialspoint.com**
Internet Source   <1%

23   **"Support Vector Machine and Long Short-term Memory using Multivariate Models for Wind Power Forecasting", International Journal of Innovative Technology and Exploring Engineering, 2020**
Publication   <1%

24   **Submitted to Liverpool John Moores University**
Student Paper   <1%

25   **"ECAI 2020", IOS Press, 2020**
Publication   <1%

26   **rip.trb.org**
Internet Source   <1%

27   **www.frontiersin.org**
Internet Source   <1%

28   **Defu Cui, Yuzhong Shen, Hong Yang, Zhitong Huang, Kyle Rush, Peter Huang, Pavle Bujanovic. "Extensible Co-Simulation Framework for Supporting Cooperative Driving Automation Research", Transportation**   <1%

Research Record: Journal of the
Transportation Research Board, 2022
Publication

29	**Submitted to University of Leeds** Student Paper	<1%
30	**Submitted to CSU, San Jose State University** Student Paper	<1%
31	**Submitted to Saint Leo University** Student Paper	<1%
32	**www.financedigest.com** Internet Source	<1%
33	**interactivechaos.com** Internet Source	<1%
34	**vocal.media** Internet Source	<1%
35	**web.archive.org** Internet Source	<1%
36	**Submitted to London School of Economics and Political Science** Student Paper	<1%
37	**dokumen.pub** Internet Source	<1%
38	**export.arxiv.org** Internet Source	<1%

mpra.ub.uni-muenchen.de

51	www.isca-speech.org Internet Source	<1%
52	www.mdpi.com Internet Source	<1%
53	www.researchgate.net Internet Source	<1%
54	xqw.co-aol.com Internet Source	<1%
55	A. KUSIAK, H. LEE*. "Neural computing-based design of components for cellular manufacturing", International Journal of Production Research, 1996 Publication	<1%

Exclude quotes   Off   Exclude matches   Off
Exclude bibliography   On