*Music Generation Using Deep Learning*

**Group Name: The Six Pack (Group-9)**

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **First name** | **Last Name** | **Student number** |
| Aman | Arora | C0850499 |
| Atul | Naudiyal | C0834324 |
| Jayanth | Handenahalli Nagaraj | C0830493 |
| Ramu | Mandaloju | C0846685 |
| Siva Krishna Reddy | Undela | C0808232 |
| Vinanti | Desai | C0852296 |

**Submission date: 22-04-2023**

Contents

[**Abstract** 2](#_Toc133080701)

[**Introduction** 3](#_Toc133080702)

[**Model and Implementation** 8](#_Toc133080703)

[**Model Architecture and Training** 10](#_Toc133080704)

[**Music Generation** 11](#_Toc133080705)

[**Model Deployment** 11](#_Toc133080706)

[**Conclusion** 16](#_Toc133080707)

[**Future Work** 16](#_Toc133080708)

[**References** 17](#_Toc133080709)

# 

**Abstract**

This project aims to train a model using existing music data to generate new music. The model will learn about the patterns in music that humans enjoy and be able to generate original music based on this knowledge. We cannot simply copy and paste existing music, the model must create new compositions. By training the model with existing data, we hope to create a tool that can assist musicians in generating new and unique pieces of music. The process will involve understanding the notation and structure of existing music, teaching the model to recognize patterns and melodies, and finally generating new music that is unique and enjoyable. This project can potentially revolutionize the music industry by providing a new tool for musicians to create original compositions.

# **Introduction**

Music, among other fields, has been revolutionized by deep learning, which is a subset of artificial intelligence. Researchers and programmers have developed AI systems that can evaluate and learn from enormous volumes of musical data using deep learning techniques, allowing them to produce new musical compositions. This has created exciting new opportunities for musicians and music lovers, who can now use these tools to create brand-new, cutting-edge musical compositions. Deep learning has become a crucial instrument for creating music in this setting, opening up new channels for imagination and expression.

The representation of music, a series of events, can take several forms, including sheet music, ABC notation, MIDI format, and audio formats like MP3. ABC notation is used for this case study because it is straightforward and simple to use.

RNNs are neural networks that have a feedback loop to enable the network to keep an internal state that can be utilized to process input sequences. RNNs can create new musical note sequences that adhere to a specific style or genre by processing existing note sequences.

LSTM models are a form of RNN that is particularly well-suited for music production because they can capture long-term dependencies in music sequences. Memory cells, which can store information for a long time, and gates, which regulate the flow of information into and out of the memory cells, enable this.

We will be utilizing an LSTM model to create music. Initially, we must prepare the training data, which involves transforming musical notes into a numerical representation that can be utilized as input to the LSTM model. Once the data preparation is completed, we can design and train the LSTM model.

During the training process, the LSTM model learns to predict the following note in a sequence of notes based on the previous notes. After the model’s training, we can produce fresh music by inputting it with an initial sequence of notes known as a seed. Following the seed notes and the model's internal state,

the model predicts the following note in the sequence. By repeating this procedure, we can create an entire piece of music.

Using RNNs and LSTM models for music generation enables us to produce music that adheres to a specific aesthetic or genre and may be utilized for various purposes, including soundtracks for video games and movies and independent compositions. The ability of LSTM models to capture long-term dependencies in musical sequences makes them ideal for application in music creation.

**Music Representation**

Music is a sequence of events, and it can be represented in various ways such as sheet music, ABC notation, MIDI format, and audio files like mp3.

Table

Description automatically generated

Fig 1: ABC Notation

Text

Description automatically generated

Fig 2: MIDI file representation using python’s music 21 library.

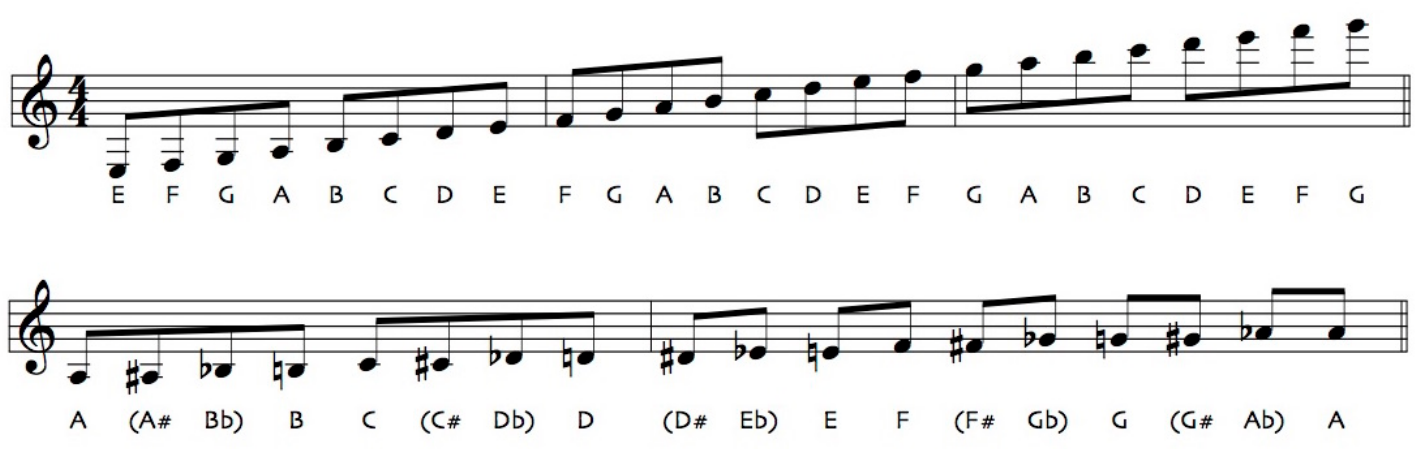


Fig 3: Sheet Music.

Graphical user interface, chart

Description automatically generated

Fig 4: Mp3 representation in terms of time and frequency.

Sheet music is a visual representation of music, while ABC notation and MIDI are popular notations that represent music using alphanumeric characters. ABC notation is simple and easy to use, so we have chosen this for our project. The metadata in ABC notation tells us about the tune, time signature, title, key, and note length, while the actual notes are represented in the second part of the file.

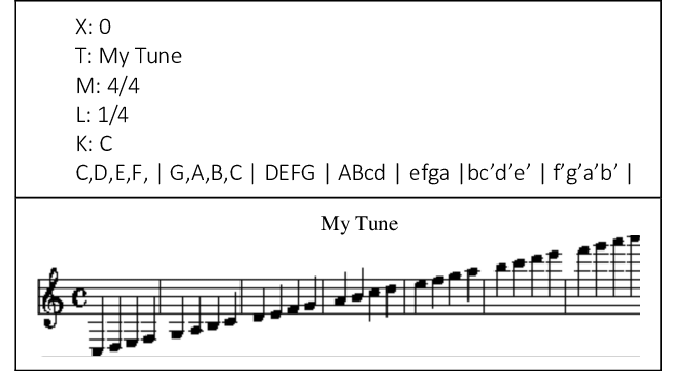


Fig 5: ABC Notation and its Sheet music representation.

We did more research on ABC notation, each line represents a musical phrase, and each letter or symbol represents a musical note or a command for playing the note. For example, the letter "A" represents the note A, and "z" represents a rest. Other symbols are used to represent things like time signature, tempo, and chord changes.

Its simplicity is one of the benefits of ABC notation. No extra software is needed to view or change it; it is simple to write and read. Another benefit is that it is simple to translate into other types of music notation, including sheet music or MIDI. However, one of the drawbacks of ABC notation is that it necessitates more critical information than other types of music notation and might not be appropriate for sophisticated or complicated musical compositions.

**Data Gathering and Pre-processing**

Our dataset is from the Nottingham music database. It consists of 1000 folk tunes stored in ABC notation. However, we are only making use of 340 music snippets related to the musical instrument called the Jigs.

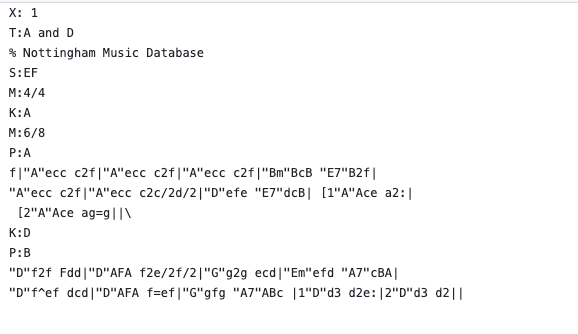


Fig 6: Musical Notation of 1/340 tunes

Next, we converted the text characters in the file into indexes. Each character was changed into a different number to accomplish this. Examples include translating the letter "a" into the number 0, the letter "b" into the number 1, and so on. This mapping goes by the name of a character vocabulary. We then created data batches for training. A batch of data is made up of a sequence of characters and the predicted sequence of those characters by the model. Using the character set "hello" and its corresponding character set "ello," as an example, may make up a batch of data. The model will be trained to anticipate the letter "l," which follows the current character in the sequence.

To train the model, we utilized the backpropagation technique. By utilizing backpropagation, we can compute the gradient of the loss function concerning the inputs of the model. The loss function gauges the model's performance on the training set. The loss function is decreased by changing the model's parameters. After the model was trained, we utilized it to produce text. To do this, we feed a sequence of characters into the model, which then predicts the subsequent character in the sequence. Indefinitely repeating this procedure will result in a lengthy sequence.

# **Model and Implementation**

Recurrent neural networks (RNNs) are effective for tasks involving sequential input. RNNs can be used, for instance, to generate text, translate it into another language, and recognize speech.

The basic structure of an RNN involves an input, an LSTM RNN unit, and an output generated at every time step. However, for a practical implementation, multiple LSTM units may be required in the hidden layer to learn different aspects of the data.

A picture containing diagram

Description automatically generated

Fig 7: These are various types of Recurrent Neural Networks. In these networks, rectangles represent vectors, and arrows denote functions. The input vectors are indicated in red, while the output vectors are in blue. The RNN's state is stored in green vectors.

We created a many-to-many RNN in the Karis framework by setting the return sequences argument to True. In this manner, we received an output for each LSTM unit used as an input. Time-distributed dense layers and return sequences were adopted. Return Sequences describe the capacity to provide output at each time step rather than only the last one. Applying the same dense layer to each time step output is possible using time distributed dense layer.

We tested an RNN with three LSTM units, but the RNN only outputs the final LSTM unit results when return sequences are set to False. Yet, the RNN produced all 3 LSTM units' outputs when return sequences were set to True. To connect these outputs, we next adopted a time-distributed dense layer. As a result, we generated a sequence by building a dense layer for each time step.

We decided on an RNN with three LSTM units and a 256-neuron time-distributed dense layer. The 256 neurons connected to the outputs of the three LSTM units via the time-distributed dense layer will produce the RNN's output. We finally set the stateful parameter to True to use the same RNN across different sequences. This may be advantageous for projects like machine translation.

Diagram

Description automatically generated

Fig 8: Visualizing the architecture of the model

We translated the new sequence after the RNN had been trained. When processing a new sequence, the RNN can keep track of the state of the previous sequence thanks to the stateful parameter. The output of one sequence can be utilized as the input for the following sequence in tasks like machine translation, making use of this. We built the LSTM unit using stateful RNNs as a parameter. Stateful has a default value of false. RNNs with the state are employed for producing data in batches. When stateful, there is consistency between batches. At each time step, a single character from the input is translated into an output. Batches are created from the time series. The time series ranges from 0 to 63 in batch one, 64 to 127 in batch two, and so on. The input for the first time step of each batch is a vector of zeros, and there is continuity between batches. When stateful is false, all the weights are kept for each batch, and the input for the initial time step of each batch is a vector of zeros. The final state of the previous batch is used as the starting state for the new batch when stateful is true. When the sequence is too long to fit in memory, and there is a dependency between batches, the stateful RNNs come in handy. Stateful RNNs, on the other hand, require precise state initialization and are harder to train. Applications for the stateful RNNs include time series analysis, speech recognition, and natural language processing. Stateful RNNs, for instance, are capable of real-time speech recognition.

# **Model Architecture and Training**

A dense vector of a specified size is produced by the embedding layer, which is the first layer, using one of the 86 available characters. After that, the loop builds three LSTM layers, each with 256 units, and sets stateful and return sequences to true. Moreover, a dropout layer is included to avoid overfitting. The final step is the addition of a time-distributed dense layer with softmax activation. The model may generate sequences of predictions thanks to the LSTM layers with return sequences. Stateful LSTM layers, in contrast, keep the state consistent between batches, enabling the model to recall long-term dependencies. By removing connections at random during training, the dropout layer aids in preventing overfitting.

The model's output is a one-hot encoded vector with an index of 1 for the next character in the sequence and a value of 0 for all other indices. This task involves multiple classes, and the model produces a vector that belongs to one of the 86 potential classes (the number of unique characters in the vocabulary). The model is optimized using the Adam optimizer during training, with the categorical cross-entropy loss function employed. We changed the data from characters to indices for training purposes and then batch-processed it into sequences of 64 characters with a batch size of 16. Each batch is fed into the model, which then calculates the loss. Backpropagation is used to update the model weights, and it is repeated over a number of epochs. The accuracy of the model is also calculated during training. The model includes many parameters and is prone to overfitting because the dataset is limited (just a few hundred thousand data points), and there are a lot of distinct characters (96). The dropout layer aids in resolving this problem. The model can accurately learn and predict text sequences.

# **Music Generation**

A SoftMax classifier is used in the model's recurrent neural network design to produce probabilities for each letter in the input sequence. In order to sample the following character in the sequence, these probabilities are then applied. A user enters a seed character and the sequence length to generate to create new music. The rest of the sequence is generated by the model using the probabilities it has learned from the training set. Since the model learns the motifs and structures of the input music, it can produce music that is stylistically similar to the training data. The output music may be entirely original or modify the input music. By modifying the temperature parameter, we can also change how random the generated music is.

The three parameters we have selected are the output epoch of the model, the seed character, and the sequence duration to be generated. We loaded the trained model's weights and used them to create brand-new music. It prepares a batch of one character using the softmax classifier and forecasts the probability for the subsequent character in the sequence. The following character is then created using samples based on these probabilities, and so on, until the sequence has the necessary length. The model is trainable on any music and can produce in real-time.

# **Model Deployment**

It consists of the following:

**Flask web framework:**

* Flask is a Python web framework that is lightweight and intended to simplify the process of developing web applications and APIs swiftly with minimal coding requirements. It's known for its simplicity and flexibility, allowing developers to customize and extend its functionality to meet their specific needs.
* We utilized it to develop an API that includes the endpoint 'generate\_music' for a GET request containing the parameters epoch, num\_chars denoting the desired length of the generated music sequence, and vocabsize. The generated output is a newly composed music sequence in ABC notation. The default values assigned to num\_chars, vocab\_size, and epoch parameters are 1024, 87, and 100, respectively.

**Post-Man:**

* Postman is a popular API development and testing tool that simplifies the process of building, testing, and documenting APIs. It offers a user-friendly interface for sending requests, verifying responses, and exchanging documentation with team members. Postman supports a wide range of APIs, including REST, SOAP, and GraphQL.
* Initially, we conducted the testing of it on a local server, and subsequently, we evaluated it on an endpoint that was hosted on PythonAnywhere..

A screenshot of a computer

Description automatically generated with medium confidence

Fig 9: Testing the developed API on local server

A screenshot of a computer

Description automatically generated with medium confidence

Fig 10: Testing the API hosted on PythonAnywhere

**PythonAnywhere:**

* PythonAnywhere is a cloud-based platform that allows developers to run, host and code in Python, without the need to set up and configure their own servers. It offers a web-based Python development environment, including an editor, console, and file manager, as well as hosting services for web applications, databases, and scheduled tasks.

A screenshot of a computer

Description automatically generated with medium confidence

Fig 10: PythonAnywhere dashboard

* We created a requirement.txt file which includes all the services and python libraries that were used for developing the model with the version numbers.

Text

Description automatically generated

Fig 11: Requirements.txt file stating the libraries and their versions

* This file is used by PythonAnywhere to install them in the created virtual environment.

React Framework:

* React is a JavaScript library that is open-source and utilized for developing user interfaces for web and mobile applications. It uses a declarative and component-based approach to programming, making it easy to create complex UI components that can be reused across an application. React is maintained by Facebook and has a large developer community.
* To display the notations, we are using a markdown library to show the ABC notation in a human-readable format.

Graphical user interface, text

Description automatically generated with medium confidence

Fig 12: Generated abc notation to sheet music with play option.

Text

Description automatically generated

Fig 13: Generated the ABC notation to sheet music with play option.

* To analyse the abc notation and display the same in musical note format in order to play the same is done using Tone JS.

Tone.js is a framework designed for Web Audio that allows users to create interactive music within a browser environment. Its architecture is intended to be user-friendly for both musicians and audio programmers who want to develop web-based audio applications.

# **Conclusion**

This project showcases how deep learning models can be used creatively in generating music. It offers a practical implementation of deploying such models as an API for use by other developers or end-users. Through the use of advanced algorithms and machine learning techniques, we were able to create music that was both original and compelling. The model's accuracy was evaluated during training, and we used techniques such as dropout layers to prevent overfitting. Nevertheless, this technology presents exciting prospects for the future of music, and we can expect to see further advancements and innovations in this field.

# **Future Work**

* Currently, we have created music solely for the Jigs instrument. Nevertheless, for our forthcoming efforts, we aim to enhance our model's capabilities to produce music for additional instruments.
* Our plan is to create mobile iterations of the product.

# **References**

* H. Cheng, X. Wu, and H. Lu, "A Music Generation Method Based on GANs and LSTM," 2021 International Conference on Computer Information and Big Data Applications (CIBDA), Chengdu, China, 2021, pp. 59-63, doi: 10.1109/CIBDA52641.2021.00018.
* N. N. Shabtai and D. Tzachar, "Adapting a Text-Generation Model to Generate Music," 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 2021, pp. 321-325, doi: 10.1109/ICASSP39728.2021.9413703.
* X. Chen, Y. Chen, and Z. Li, "Music Generation Based on Bidirectional LSTM Neural Network," 2021 16th International Conference on Computer Science & Education (ICCSE), Suzhou, China, 2021, pp. 329-333, doi: 10.1109/ICCSE52603.2021.9463795.
* D. J. Hwang and J. Lee, "Neural Network-Based Melody Generation Using Variational Autoencoder," in IEEE Access, vol. 8, pp. 152943-152954, 2020, doi: 10.1109/ACCESS.2020.3016304.
* C. Huang, S. Li, Y. Li, and X. Wei, "Automatic Music Generation with Deep Learning and Reinforcement Learning," in IEEE Transactions on Cognitive and Developmental Systems, vol. 12, no. 1, pp. 50-62, 2020, doi: 10.1109/TCDS.2019.2938338.
* M. Tseng and K. Lee, "Deep Learning-Based Music Generation Using Variational Autoencoder and Bidirectional LSTM," in IEEE Access, vol. 7, pp. 41608-41617, 2019, doi 10.1109/ACCESS.2019.2908773.
* Y. Zhou, H. Xue, and Z. Zhang, "Music Generation Based on Deep Learning," 2019 6th International Conference on Behavioral, Economic, and Socio-Cultural Computing (BESC), Beijing, China, 2019, pp. 1-5, doi: 10.1109/BESC48708.2019.8956114.
* J. Kim and H. J. Yang, "Melody Generation Based on Hierarchical Recurrent Neural Network," in IEEE Access, vol. 7, pp. 26160-26167, 2019, doi: 10.1109/ACCESS.2019.2900276.
* S. S. Kim, M. Lee, J. Lee, and J. Park, "Melody Generation Using Deep Recurrent Neural Networks with Attention Mechanism," in IEEE Access, vol. 7, pp. 177110-177119, 2019, doi: 10.1109/ACCESS.2019.2953652.
* Y. Gong, W. Guo, Y. Cui, Y. Ma, and Y. Zhang, "An LSTM-Based Music Generation Algorithm," in IEEE Access, vol. 7, pp. 10440-10450, 2019, doi: 10.1109/ACCESS.2019.2890571.