Automated Music Generation using Deep Learning

Shikhar Bhardwaj
MSc. Data Science
University of Europe for Applied Sciences
Berlin, Germany
Shikhar.Bhardwaj@ue-germany.de

Shalmiya Mundeth Salim
MSc. Data Science
University of Europe for Applied Sciences
Berlin, Germany
Shalmiya.Salim@ue-germany.de

Dr. Talha Ali Khan
MSc. Data Science
University of Europe for Applied Sciences
Berlin, Germany
talhaali.khan@ue-germany.de

Soudeh JavadiMasoudian
MSc. Data Science
University of Europe for Applied Sciences
Berlin, Germany
Soudeh.javadimasoudian@ue-germany.de

2022 International Conference Automatics and Informatics (ICAI) | 978-1-6654-7625-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAI55857.2022.9960063

***Abstract*— Generating music using Deep Neural Networks has been an active field of study for the past few decades. This paper discusses how to create computer-based monophonic musical content using a Recurrent Neural Network with the extension of Long Short-Term Memory. The objective of this study is to generate music that has a melodic nature to it and can be played by humans using sheet notation. The neural network is fed with input from the video game series: Final Fantasy for creating the melody. The selection of instruments is piano, which will be the same for the input and output. The main focus of this paper is on constrained music, differentiating notes from chords to help the architecture learn to distinguish one from the other and generate music with a mix of both, thus creating a pleasant listening experience.**

Keywords—octave, LSTM, Feature Extraction, Encoding, Neural Network Design

I. Introduction

Writing music is an art that requires knowledge of music theory and a creative mindset. Music theory consists of single notes and chords in their most basic form. To progress from an individual note to a chord, multiple notes are played at the same time. Sheet notation is the most commonly used method of writing music with notes and chords. The desired output for this study is a sheet notation of a musical piece entirely composed by a computer that musicians can play. Considering the complexity of music composition, the generated music supports musicians in the song creation process and inspires artists in phases of creative droughts. Furthermore, individuals without prior knowledge of music theory can use the output as an instrumental track for voiceovers and other creative endeavours. Finally, when played on its own, the musical piece should showcase melodic sequences with changing notes and chords to create a pleasant listening experience for the listener.

Section II covers a literature review for selecting the deep neural network, assessing different types of architectures with regard to their suitability for music generation. In terms of project implementation, Section III offers a detailed description of the approach. This section contains information on which input data set was used, what requirements the user should consider for code execution, and the entire script to run the code. Section IV assesses the results of our paper before discussing concluding remarks in Section V with study limitations and potential future research avenues.

II. Literature Review

A. Feedforward Neural Network

Feedforward Neural Networks can be described as the simplest form of an artificial neural network. Each Feedforward Neural Network consists of an input layer, an output layer and one or more hidden layers.

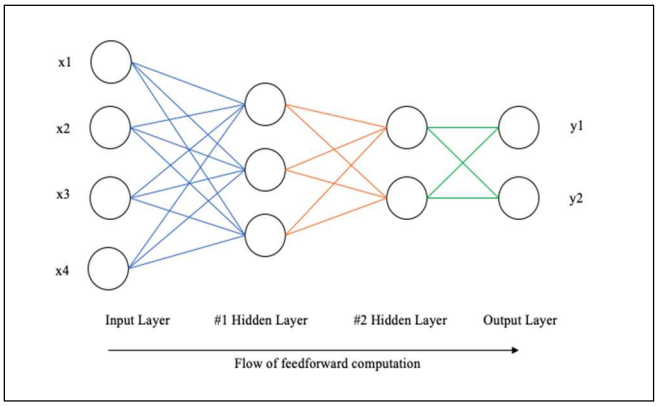

Fig. 1. An example of a Feedforward Neural Network

The network's depth is determined not by the total number of layers but only by the number of weighted layers.

An example of a three-layered Feedforward Neural Network with two hidden layers can be seen in Fig. 1. The flow of computations within this type of architecture is one-way. Whereas the input layer is the starting point of the data flow as the first layer, the output layer is the last layer collecting the generated output of the network. Every layer contains nodes connected with every single node from the previous layer. However, there is no connection between nodes of the same layer. The connection strength between the nodes can vary according to their respective weights calculated in a weight matrix.

B. Recurrent Neural Network

Recurrent Neural Networks are similar to Feedforward Neural Networks with regard to the setup of the architecture but with a critical difference. Recurrent Neural Networks are recursive because of the recurrent connections within the hidden layers. Therefore, the flow of data is not linear, as is the case in Feedforward Neural Networks. Instead, it has a

978-1-6654-7625-6/22/$31.00 ©2022 IEEE

recursive data structure as the output of the hidden layer re-enters the hidden layer with a corresponding weight matrix as an additional input in addition to the inputs received from the input layer. An example of a three-layered RNN, including two hidden layers, can be seen in Fig. 2.

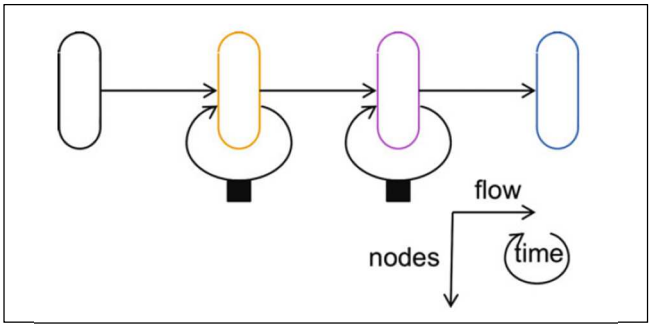

Fig. 2. An example of a Recurrent Neural Network

This type of architecture is ideal for music generation as it uses sequential information. Unlike the Feedforward Neural Networks, where the output is independent of previous computations, Recurrent Neural Networks can utilise the architecture's recursive nature to predict the following note in a sequence of notes. In order to achieve an even better output, the RNN model is extended with Long Short-Term Memory. LSTM, as an extension of RNN, helps create a model that recognises and encodes long-term patterns which are noticeably prevalent in music. LSTMs store information within memory cells, which are shielded from the standard data flow of the RNN. This property allows them to remember useful long-term correlations, unlike the standard RNN, which suffers from the vanishing or exploding gradient problem.

III. IMPLEMENTATION

This section will explain the selection and pre-processing of data, the process of training the network, and the generation of instrumental music compositions. Experiments were performed on the Amazon Web Server Platform with deep learning implementation done in Keras and TensorFlow.

A. Tools Used

The following list outlines the different types of tools used for the implementation of the proposed Recurrent Neural Network:

- Python: The proposed neural network uses Python as it offers concise and readable code.
- Music21: Python codes from the Music21 toolkit have been applied to convert the MIDI files into music notations during this study's encoding and decoding section. It is a computer-assisted musicology tool that helps to understand music fundamentals and create a MIDI output from our neural network.
- Keras: In the proposed methodology, Keras has been used to build and train the LSTM model, which has been used to generate the musical notation.
- H5Py: Since this study deals with numerous music objects, H5Py has been used in this network to store and manipulate a vast quantity of data from NumPy.

B. Data Preparation

This section includes an explanation regarding the data gathering process for the model, further processing of the data to use in the LSTM model, and how this network model is constructed.

- Dataset: Piano music from Final Fantasy soundtracks has been chosen as the dataset of this study. This is because of most of the pieces' unique and pleasing melodies and the perfect number of components that exist. The dataset contains 92 songs with 352 different notes and chords.
- Examining the data: The first step in analysing the data is identifying the elements of the MIDI files. Fig. 3 shows an extract from a MIDI file that has been read by Music21.

There are two object types of data: Notes and Chords. Chord objects represent a container for a set of notes that are played simultaneously, producing a distinct sound as a group. Each Note object contains pitch, octave, and offset information.

Pitch refers to the frequency of the sound, or its amplitude, i.e., how high or low it is. Music is made up of seven notes, and each note has a letter represented by the letters A, B, C, D, E, F, G, and A is the highest note in music, while the lowest is G.

```
...
<music21.note.Note F>
<music21.chord.Chord A2 E3>
<music21.chord.Chord A2 E3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.note.Note G>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord A2 E3>
...
```

Fig. 3. MIDI file read by Music21 Library

Octave refers to which set of pitches you use on a piano. In music, it's an interval whose higher note has a sound-wave frequency of vibration twice that of its lower note.

Offset refers to where the note is located in the piece.

C. Feature Extraction and Encoding

For the neural network to predict a musical piece or which note or chord is next, it should have all the chords and note objects of the training data set in its prediction array. This step extracts those notes and chords and converts them into network input and output for the proposed LSTM network.

In the beginning, the input data needs to be loaded into an array, as shown in Fig. 4. The function used in this study to convert the loaded data into notes and chords from the Music21 stream object is a *converter. parse(file).* In this process, at first, append the pitch of every object using its string notation as it is possible to recreate the important part of the note using the string notation of the pitch. Encode the id of each note in the chord into a single string, separated by a dot between each note, to append the chord. Encodings like these enable us to easily decode the output from the neural network into the right notes and chords.

D. Network Input and Output

As the first step of compiling all the notes and chords into a sequential list has been done, the next stage of data preparation is to prepare the input-output sequences for the proposed neural network.

```
from music21 import converter, instrument, note, chord

notes = []

for file in glob.glob("midi_songs/*.mid"):
    midi = converter.parse(file)
    notes_to_parse = None

    parts = instrument.partitionByInstrument(midi)

    if parts: # file has instrument parts
        notes_to_parse = parts.parts[0].recurse()
    else: # file has notes in a flat structure
        notes_to_parse = midi.flat.notes

    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in
element.normalOrder))
```

Fig. 4. MIDI file feature extraction

- **Mapping**: As the sequences of notes and chords are in a categorical data form, it is essential to convert them into an integer-based format to get the maximal performance of a neural network. This study has used a mapping function for performing this task shown in Fig 4.
- **Input/output creation**: This study used the Python library's range and appended functions to create the network output. The length of the input sequence chosen is 100; as a result, the prediction of the next note can be made based on the 100 notes that preceded it. In other terms, the network output of each input sequence would be the first note or chord that comes after the sequence of notes in the input sequence in the list of notes. The programmatic depiction of this part is also shown in Fig. 5.

```
sequence_length = 100

# get all pitch names
pitchnames = sorted(set(item for item in notes))

# create a dictionary to map pitches to integers
note_to_int = dict((note, number) for number, note in
enumerate(pitchnames))

network_input = []
network_output = []

# create input sequences and the corresponding outputs
for i in range(0, len(notes) - sequence_length, 1):
    sequence_in = notes[i:i + sequence_length]
    sequence_out = notes[i + sequence_length]
    network_input.append([note_to_int[char] for char in
sequence_in])
    network_output.append(note_to_int[sequence_out])

n_patterns = len(network_input)
```

Fig. 5. Mapping and input-output sequences

- **Normalisation and One-hot encoding**: The function used for reshaping the network input into a format compatible with the LSTM layers is *NumPy.reshape.* In the next step, normalise these integer data inputs to fit a probability distribution using the *float* function. Finally, a one-hot encoding process is done for the output data to represent it in a binary vector format using *np_utils.to categorical* function. Binary vector data would support the deep learning algorithm to improve the prediction. Fig. 6 shows the representation of the above three processes.

```
network_input = numpy.reshape(network_input, (n_patterns,
sequence_length, 1))
# normalize input
network_input = network_input / float(n_vocab)

network_output = np_utils.to_categorical(network_output)
```

Fig. 6. Normalisation and One-hot encoding

E. Training: Deep Neural Network Design

This section will explain the proposed model architecture and the process for training the extracted features into a prediction model.

So far, this paper has covered the first three processes in the architecture as a part of training data preparation, and the next step is to design a neural network for music data prediction. The proposed architecture flow is shown in Fig. 7.

The neural network is designed primarily using three network layers: the LSTM layer, the Dropout layer, and the Dense layer.

a) LSTM Layer: Long Short-Term Memory networks are recurrent neural networks capable of learning order dependence in sequence prediction problems. Unlike the standard recurrent neural network, this layer has a forget cell in each neuron that can recognise and encode long-term patterns. This is why LSTM-based RNN has been chosen in this study over normal RNN. Generally, it uses the gradient descent algorithm to correct weights across the network for accurate prediction. Three LSTM layers have been used in this architecture, as shown in Fig. 8.

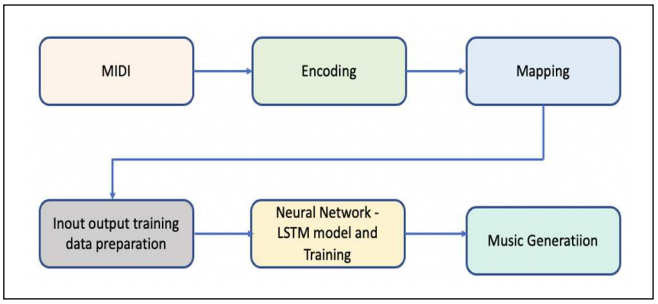

Fig. 7. Proposed Architecture

b) Dropout Layer: This layer is added to the network to prevent the problem of overfitting and thereby rule out the option of generating a music output that sounds substantially similar to the input data. A fraction of input data will be dropped while transiting through the Dropout layer, and this fraction is defined by the parameter used with the layer. Three Dropout layers have been used in the proposed architecture.

c) Dense Layer: It is a fully connected layer; each neuron in the input layer or precedent layer would be connected to all the neurons in the next layer and generates one output per neuron. This layer is added to classify the features with more accuracy. Two Dense Layers are included in the network design.

d) Activation Layer: This layer decides whether a neuron should be activated or not and, based on which, generates output. This study uses a rectified linear unit (ReLu) as an activation function.

e) Loss Function: It quantifies the difference between the expected and neural network outcomes. This study will evaluate how well the algorithm models the input data. This study aims to reduce the loss function to a small value to get a nearly accurate output. In this network design, Categorical cross-entropy has been used with softmax activation function as loss function of the neural network.

f) Optimiser: The function of the optimiser in the deep neural network layer is to fine-tune the attributes of the network, such as weights and learning rate. This study has adopted the RMSprop optimiser, which uses the RMSprop algorithm to optimise the loss function and improve the accuracy.

g) Batch Normalisation Layer: This study has used the batch normalisation layer of Keras in addition to other layers to automatically standardise the inputs to a layer in the neural network. It has a regularisation function similar to the dropout layer.

Fig. 8 shows the overall architecture of the neural network. Although the data flow would not necessarily be in an exact downward direction from layer to layer, it would provide a general idea about the network elements.

The next step is to arrange these layers in the network model using the Keras library. The first parameter mentioned in each layer defines the number of nodes each layer has. LSTM layer in this network comprises 512 nodes, 256 nodes for the Dense layer and Dropout layer, and 0.3 fractions of inputs would be set to zero to prevent the overfitting of input data in the model.

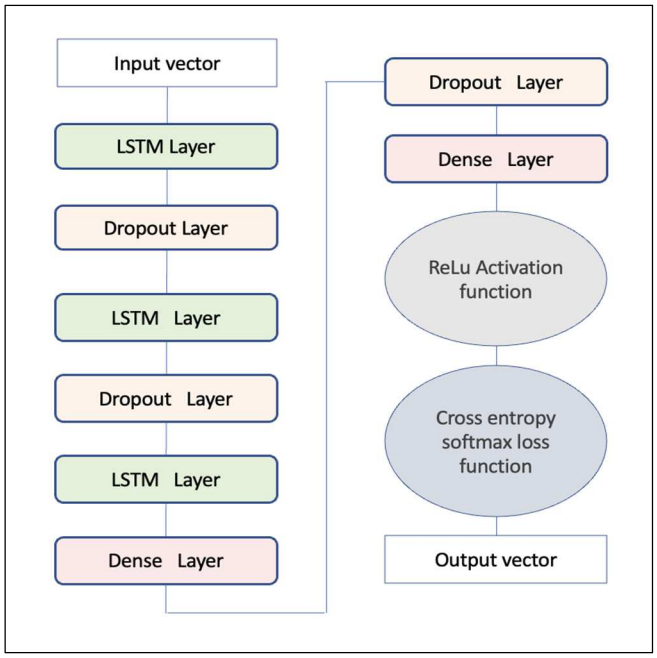

Fig. 8. Neural Network Design

To inform the network about the shape of the training data, a unique parameter has been defined in the network's first layer, namely *input_shape*. Neural network design using the Keras library is given in Fig. 9.

```
model = Sequential()
  model.add(LSTM ( 512,
  input_shape=(network_input.shape[1], network_input.shape[2]),
  recurrent_dropout=0.3,
  return_sequences=True
  ))
  model.add(LSTM(512, return_sequences=True, recurrent_dropout=0.3,))
  model.add(LSTM(512))
  model.add(BatchNorm())
  model.add(Dropout(0.3))
  model.add(Dense(256))
  model.add(Activation('relu'))
  model.add(BatchNorm())
  model.add(Dropout(0.3))
  model.add(Dense(n_vocab))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Fig. 9. Neural network design using Keras

The nodes in the network's last layer would be the same as several system outputs so that they will directly map to the output classes.

Further to the definition of the network, the next task is to give instructions to the training network with regard to the number of iterations or epochs it needs to be performed. This neural network has been set to process 200 epochs in the network to optimise the weights. The model. fit function is used to train the network. Furthermore, a model checkpoint is defined to save the weights to a file after each epoch so that data is not lost in an unexpected situation. It can also be used to stop the iterations at a particular timestamp based on the optimisation of the weights so far. The final output from the training network would be the optimised weights file called weights.hdf5, which will be used for data prediction. Fig. 10 shows the program for performing the above-defined tasks.

```
filepath = "weights-improvement-{epoch:02d}-{loss:.4f}-bigger.hdf5"

checkpoint = ModelCheckpoint(
    filepath, monitor='loss',
    verbose=0,
    save_best_only=True,
    mode='min'
)
callbacks_list = [checkpoint]

model.fit(network_input, network_output, epochs=200, batch_size=64,
callbacks=callbacks_list)
```

Fig. 10. Training the network and checkpoint in the model

F. Generating Music

The network would be ready to produce the music once it has been successfully trained. To predict the data, it should start from the same state, and for simplification purposes, this study has used the network design code from the training part, as in Figure 11. In addition, the weights.hdf5 file needs to be loaded using the *model. weights* python function so that the proposed network would use the optimised weights for prediction.

```
start = numpy.random.randint(0, len(network_input)-1)

int_to_note = dict((number, note) for number, note in
enumerate(pitchnames))

pattern = network_input[start]
prediction_output = []

# generate 500 notes
for note_index in range(500):
    prediction_input = numpy.reshape(pattern, (1, len(pattern), 1))
    prediction_input = prediction_input / float(n_vocab)

    prediction = model.predict(prediction_input, verbose=0)

    index = numpy.argmax(prediction)
    result = int_to_note[index]
    prediction_output.append(result)

    pattern.append(index)
    pattern = pattern[1:len(pattern)]
```

Fig. 11. Prediction and decoding using neural network

This study has used NumPy.*random.randint* function to start the prediction process from a random index point so that a different output would be obtained each time corresponding to the input data loaded. The network is set to generate 500 notes to create a two minutes music piece and use NumPy.*argmax* prediction function in the model.

The final step is decoding the integer-based numerical data to notes and converting it into a MIDI file.

As shown in Fig. 12, when submitting the sequence for music generation, the first sequence would always be the one from starting index; for the next sequence, the network will replace the last note of the sequence with the output node of the first sequence after iteration. The process repeats until it generates 500 notes from the prediction output array. In this example shown in Fig. 12, the output of the first iteration is F.

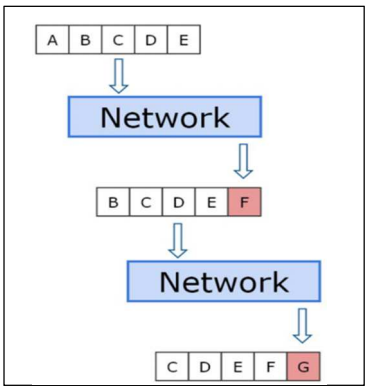

Fig. 12. Input sequence format during prediction

In the next phase, map each network output with the note classes by comparing the probability of each output note and will accept the output with the highest probability index as the most likely prediction of this network. In this case, shown in Fig. 13, class D is the predicted output with the highest probability.

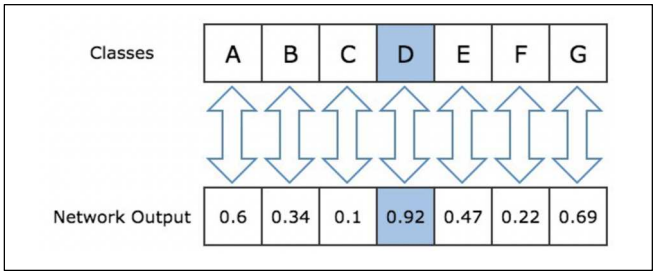

Fig. 13. Output prediction using Probability Index

Subsequently, all the outputs are collected into a single array, and specific steps must be followed to decode the array of notes and chords into music21 notes and chord objects. For a chord pattern, first, it will be separated into an array of notes, after which, using a loop of the string representation of the note, create a note object per each note which can be combined afterwards to create a chord object. To decode a note pattern to a note object, repeat the preceding steps until the note object is created. Both processes are explained programmatically in Fig. 14.

```
for pattern in prediction_output:
    # pattern is a chord
    if ('.' in pattern) or pattern.isdigit():
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            new_note = note.Note(int(current_note))
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        output_notes.append(new_chord)
    # pattern is a note
    else:
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        output_notes.append(new_note)
```

Fig. 14. Note and Chord pattern to objects conversion

Finally, list these outputs to a music stream object and write it to a MIDI file using the write function in the Music21 toolkit shown in Fig. 15.

```
midi_stream = stream.Stream(output_notes)

midi_stream.write('midi', fp='test_output.mid'
```

Fig. 15. MIDI file creation

IV. Analysis of Results

Through this study, it was possible to understand and explore the behaviour of LSTM RNN to generate music by following an approach entailing data preparation, neural network training, and music generation. Although the music generated through this study using this neural network was not magnificent when compared with the input music data, it had some structure and a basic understanding of notes and chords. Some of the problems that remain unsolved are the inclusion of more music stream objects and experimenting with some more parameters of music, such as offset.

V. Conclusions

Automated sequence generation using LSTM RNNs is a promising approach. They solve problems by making decisions based on context, which allows them to predict the sequence's next member. Using this method, it is possible to successfully generate a piece of monophonic music that is entirely created by a computer and requires no human involvement or prior knowledge of music theory.

A. Future Research Avenues

There is further scope, although this study has created a decent piece of music using the LSTM method.

First, the model can be trained with more tunes, creating a more distinct and unique piece of melody. Furthermore, this study has not experimented with the offset of note, i.e., where the note is located in a music piece, and also experimenting with more classes would give more scope for improvement.

Secondly, this study only focused on monophonic music creation; manipulating the network using polyphonic music would also improve the learning process in creating a more exciting melody music piece.

Third, it is possible to create a robust network by adding more layers to the network and increasing the iterations for the optimisation of weights in a more accurate way. This process requires a system with strong computing power and more space.

Finally, a method can be added to the model to manage the unknown notes in the music by replacing them with the known notes. It can also add the starting and ending music in every network output to generate more standard and melodious music pieces nearly identical to one created by humans.

References

[1] Jean-Pierre Briot, Gaetan Hadjeres and Franc¸ois-David Pachet, "Deep Learning Techniques for Music Generation," Computational Synthesis and Creative Systems, Springer, 2019,

[2] Sigurður Skúli, “How to generate Music using an LSTM Neural Network in Keras”. https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5 (accessed July 10. 2022).

[3] User's Guide, Chapter 4: Lists, Streams (I) and Output — music21 Documentation.Available:https://web.mit.edu/music21/doc/usersGuide/usersGuide_04_stream1.html#usersguide-04-stream1 (accessed July 10. 2022).

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

[5] Ronen Eldan and Ohad Shamir. "The power of depth for feedforward neural networks", May 2016.

[6] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks. Neural Networks", 4(2):251–257, 1991.

[7] Douglas Eck and J urgen Schmidhuber. "A first look at music composition using LSTM recurrent neural networks". Technical report, IDSIA/USI-SUPSI, Manno, Switzerland, 2002. No.IDSIA-07-02.

[8] Alex Graves. "Generating sequences with recurrent neural networks", June 2014.

[9] Keunwoo Choi, Gy orgy Fazekas, and Mark Sandler, "Text-based LSTM networks for automatic music composition". 1st Conference on Computer Simulation of Musical Creativity (CSMC 16), Huddersfield, U.K., June 2016.

[10] Feynman Liang. BachBot: Automatic composition in the style of Bach chorales – Developing, analysing, and evaluating a deep LSTM model for musical style. Master's thesis, University of Cambridge.

[11] Tianyu Jiang and Qinyin Xiao, "Music Generation Using Bidirectional Recurrent Network", 2019 IEEE 2nd International Conference on Electronics Technology.

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

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

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

[15] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE Transactions on Signal Processing 45.11 (1997): 2673-2681.

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

[17] Eck D, Schmidhuber J. A first look at music composition using lstm recurrent neural networks[J]. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2002, 103